PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Brenno dos Santos Neves

ANÁLISE DE PROGRAMAS DE SERVIÇOS DE STREAMING

Vitória ES
2023

Brenno dos Santos Neves

ANÁLISE DE PROGRAMAS DE SERVIÇOS DE STREAMING

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Vitória ES

2023

SUMÁRIO

1. Introdução

1.1. Contextualização

O projeto consiste em uma análise dos programas de streaming das empresas Netflix e Amazon, os programas serão classificados entre “Bom” e “Ruim” com base na pontuação do IMBD, quantidade de visualizações e quantidade de votos. Tentaremos ver se tem alguma relação entre a pontuação do IMDB com o TMDB se as 2 grandezas estão relacionadas e classificar se os programas são "Bom" ou “Ruim”.

1.2. O problema proposto

Para criar um modelo de machine Learning que indique quais os programas podem ser classificados entre bons e ruins.

Os dados foram pegos do Kaggle, baseados nos serviços de streaming da Netflix e da Amazon Prime vídeo, conforme a classificação abaixo:

```
# Classificação nos gostos do usuário BRENNO
#imdb >= 7 a pontuação tmdb >=6 o número de runtime maior que 70 e quantidade de votos deve ser maior que 400
filtro = [
    (dataSet['imdb_score'] >= 7) & (dataSet['tmdb_score'] >= 6) & (dataSet['runtime'] >= 70) & (dataSet['imdb_votes'] >= 400)
]

#Resultados
resultado = ['Bom']

#Novo coluna com o status de bom e 0
dataSet['Status'] = np.select(filtro, resultado)

#Tudo que for 0 substitui para Ruim
dataSet['Status'] = dataSet['Status'].str.replace("0",'Ruim')

#Verificar se todos os dados estão divididos entre "BOM" e "RUIM"
dataSet['Status'].unique()
```

```
array(['Ruim', 'Bom'], dtype=object)
```

Os programas com pontuação IMDB acima ou igual a 7, pontuação do TMDB igual ou superior a 6, quantidade de runtimes igual ou superior a 70 e quantidade de votos do IMDB igual ou superior a 400.

Os dados que temos são:

Netflix

```
dfNetflix
```

	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb_votes	tmdb_score	streaming
1	Taxi Driver	MOVIE	1976	114	['drama', 'crime']	['US']	8.2	808582.0	8.179	Netflix
2	Deliverance	MOVIE	1972	109	['drama', 'action', 'thriller', 'european']	['US']	7.7	107673.0	7.300	Netflix
3	Monty Python and the Holy Grail	MOVIE	1975	91	['fantasy', 'action', 'comedy']	['GB']	8.2	534486.0	7.811	Netflix
4	The Dirty Dozen	MOVIE	1967	150	['war', 'action']	['GB', 'US']	7.7	72662.0	7.600	Netflix
5	Monty Python's Flying Circus	SHOW	1969	30	['comedy', 'european']	['GB']	8.8	73424.0	8.306	Netflix
...	...	...	...	...	...	...	...	...	...	...
5838	Happiness Ever After	MOVIE	2021	99	['drama', 'romance']	['ZA']	4.2	163.0	7.300	Netflix
5842	Super Monsters: Once Upon a Rhyme	MOVIE	2021	25	['animation', 'family']	[]	5.6	38.0	6.300	Netflix
5843	My Bride	MOVIE	2021	93	['romance', 'comedy', 'drama']	['EG']	5.0	327.0	5.300	Netflix
5847	Lokillo	MOVIE	2021	90	['comedy']	['CO']	3.8	68.0	6.300	Netflix
5849	Mighty Little Bheem: Kite Festival	SHOW	2021	7	['family', 'animation', 'comedy']	[]	7.8	18.0	10.000	Netflix

Amazon Prime Video

```
dfAmazon
```

	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb_votes	tmdb_score	streaming
0	The Three Stooges	SHOW	1934	19	['comedy', 'family', 'animation', 'action', 'f...	['US']	8.6	1092.0	7.6	Amazon
1	The General	MOVIE	1926	78	['action', 'drama', 'war', 'western', 'comedy'...	['US']	8.2	89766.0	8.0	Amazon
2	The Best Years of Our Lives	MOVIE	1946	171	['romance', 'war', 'drama']	['US']	8.1	63026.0	7.8	Amazon
3	His Girl Friday	MOVIE	1940	92	['comedy', 'drama', 'romance']	['US']	7.8	57835.0	7.4	Amazon
4	In a Lonely Place	MOVIE	1950	94	['thriller', 'drama', 'romance']	['US']	7.9	30924.0	7.6	Amazon
...	...	...	...	...	...	...	...	...	...	...
9843	Ammaa Ki Boli	MOVIE	2021	117	['comedy', 'drama']	['IN']	7.3	1335.0	1.0	Amazon
9844	Alleyway	MOVIE	2021	67	['action', 'crime', 'thriller']	[]	5.4	92.0	6.8	Amazon
9847	Girls' Night In	MOVIE	2021	91	['comedy', 'drama']	['US']	2.8	28.0	7.0	Amazon
9856	Anbirkiniyal	MOVIE	2021	118	['thriller', 'drama']	['IN']	6.8	361.0	7.0	Amazon
9864	Gun and a Hotel Bible	MOVIE	2021	58	['drama']	[]	4.0	142.0	6.5	Amazon

HBO

dfHbo

	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb_votes	tmdb_score	streaming
0	The Wizard of Oz	MOVIE	1939	102	['fantasy', 'family']	['US']	8.1	389774.0	7.6	HBO
1	Citizen Kane	MOVIE	1941	119	['drama']	['US']	8.3	433804.0	8.0	HBO
2	Casablanca	MOVIE	1942	102	['drama', 'romance', 'war']	['US']	8.5	558849.0	8.2	HBO
3	The Big Sleep	MOVIE	1946	116	['thriller', 'crime']	['US']	7.9	84494.0	7.7	HBO
4	The Maltese Falcon	MOVIE	1941	100	['thriller', 'romance', 'crime']	['US']	8.0	156603.0	7.8	HBO
...	...	...	...	...	...	...	...	...	...	...
3275	Breathless	MOVIE	2021	106	['crime', 'drama', 'thriller']	['DO']	6.3	27.0	5.9	HBO
3279	Furry Friends Forever: Elmo Gets a Puppy	MOVIE	2021	26	['animation']	['US']	6.8	14.0	10.0	HBO
3283	Marlon Wayans: You Know What It Is	MOVIE	2021	58	['comedy']	['US']	3.8	224.0	5.4	HBO
3284	Ahir Shah: Dots	MOVIE	2021	61	['comedy']	[]	5.8	69.0	7.0	HBO
3290	Algo Azul	MOVIE	2021	90	['comedy']	['PA']	5.9	50.0	2.0	HBO

1.3. Objetivos

Criar uma modelo para classificar os programas dos serviços de streaming entre “Bom” e “Ruim”. Verificar se existe alguma relação entre as grandezas IMDB e TMDB, que serão usadas para classificar os filmes também, junto com a quantidade de execuções e a quantidade de votos.

2. Coleta de Dados

Para o tratamento do problema proposto, foram utilizados 3 datasets sendo 2 de para análise e treinamento do modelo e o terceiro para testes de nova carga, os dados foram extraídos do site kaggle.

Os links dos 3 datasets usados estão abaixo:

- https://www.kaggle.com/datasets/shivamb/amazon-prime-movies-and-tv-shows
- https://www.kaggle.com/datasets/victorsoeiro/hbo-max-tv-shows-and-movies
- https://www.kaggle.com/datasets/victorsoeiro/netflix-tv-shows-and-movies

Todos os 3 datasets foram filtrados para terem as mesmas variáveis.

Nome da coluna/campo	Descrição	Tipo
title	Título do programa	object
type	Tipo de programa	object
release_year	Ano de lançamento	Int64
runtime	Quantidade de Execuções	Int64
genres	Gênero	object
production_countries	Países que produziram	object
imdb_score	Pontuação do IMDB	Float64
imdb_votes	Quantidade de votos da plataforma IMDB	Float64
tmdb_score	Pontuação da TMDB	Float64
streaming	Serviço que reproduz o programa	object

```
dataSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8296 entries, 1 to 9864
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   title                 8296 non-null   object
 1   type                  8296 non-null   object
 2   release_year          8296 non-null   int64
 3   runtime               8296 non-null   int64
 4   genres                8296 non-null   object
 5   production_countries  8296 non-null   object
 6   imdb_score            8296 non-null   float64
 7   imdb_votes            8296 non-null   float64
 8   tmdb_score            8296 non-null   float64
 9   streaming             8296 non-null   object
dtypes: float64(3), int64(2), object(5)
memory usage: 712.9+ KB
```

3. Processamento/Tratamento de Dados

A importação dos dados, o processamento, o tratamento e a modelagem dos dados foram todos executados no Anaconda, usando o serviço do Jupyter Notebook versão 6.4.12, usando a linguagem python na versã0 3.9.13.

Também foi usado as bibliotecas Pandas, Numpy, Datetime, Matplotlib, Seaborn, Plotly, Statsmodels, Scipy, Sklearn, Apyori e Graphviz

Importação das bibliotecas

```
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import statsmodels
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from statsmodels.stats.diagnostic import lilliefors
from apyori import apriori

# Modelo de Naive Bayes
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
from yellowbrick.classifier import ConfusionMatrix

#Modelo de Arvore de decisão
from sklearn.tree import DecisionTreeClassifier
import graphviz
from sklearn.tree import export_graphviz

# Seleção de Atributos
from sklearn.svm import SVC
from sklearn.ensemble import ExtraTreesClassifier

# Random Forest
from sklearn.ensemble import RandomForestClassifier
```

Depois foi realizado a carga dos três datasets pelos comandos abaixos:

```
# Carregar os dados das operadoras de Streaming
dNetflix = pd.read_csv('PUC/Netflix.csv')
dAmazon = pd.read_csv('PUC/amazon_prime.csv')
dHbo = pd.read_csv('PUC/HBO.csv')
```

Foi realizado como tratamento a limitação dos campos que serão deixados nos datasets para serem usados nas análises.

Conforme explicado acima, todos trouxeram os mesmos campos, definidos entre title, type, release_year, runtime, genres, production_countries, imdb_score, imdb_votes e tmdb_score.

Foram removidos os dados vazios de todos os datasets.

```
# Selecionar quais campos iremos usar das 3 bases carregadas
dfNetflix = dNetflix[['title','type','release_year','runtime','genres','production_countries','imdb_score',
'imdb_votes','tmdb_score']]
dfNetflix= dfNetflix.dropna()
dfNetflix['streaming'] = 'Netflix'
dfAmazon=dAmazon[['title','type','release_year','runtime','genres','production_countries','imdb_score',
 'imdb_votes','tmdb_score']]
dfAmazon= dfAmazon.dropna()
dfAmazon['streaming'] = 'Amazon'
dfHbo=dHbo[['title','type','release_year','runtime','genres','production_countries','imdb_score',
'imdb_votes','tmdb_score']]
dfHbo = dfHbo.dropna()
dfHbo['streaming'] = 'HBO'
```

O dataset da HBO foi preciso padronizar ele, separado dos outros, pois os dados da Netflix e da Amazon seriam concatenados em um único dataset para depois sofrerem uma limpeza dos caracteres especiais, como o do HBO seria usado como uma carga de teste de um dos modelos, ele foi carregado e mantido separado e com isso teve que receber um tratamento de dados separadamente.

Carregar o data set da HBO como carga de teste

```
# Nova carga de teste
dfNovaCarga = dHbo[["title","type","release_year","runtime","genres","production_countries","imdb_score","imdb_votes",
                    "tmdb_score"]]
# Remover caracter [ ]
# Remover linhas com indices duplicados
dfNovaCarga = dfNovaCarga[~dfNovaCarga.index.duplicated()]
# Iremos remover as linhas cujo o pais seja vazio
dfNovaCarga = dfNovaCarga[dfNovaCarga.production_countries !='']
# Remover aspas simples das colunas genres e production_countries
dfNovaCarga["genres"] = dfNovaCarga["genres"].str.replace("'", "")
dfNovaCarga["production_countries"] = dfNovaCarga["production_countries"].str.replace("'", "")

dfNovaCarga["genres"] = dfNovaCarga["genres"].str.replace("]","", regex=True)
dfNovaCarga["genres"] = dfNovaCarga["genres"].str.replace("[","", regex=True)
dfNovaCarga["production_countries"] = dfNovaCarga["production_countries"].str.replace("]","", regex=True)
dfNovaCarga["production_countries"] = dfNovaCarga["production_countries"].str.replace("[","", regex=True)
dfNovaCarga = dfNovaCarga.dropna()
# Iremos remover as linhas cujo o pais seja vazio
dfNovaCarga = dfNovaCarga[dfNovaCarga.production_countries !='']
dfResult=dfNovaCarga
dfResult
```

	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb_votes	tmdb_score
0	The Wizard of Oz	MOVIE	1939	102	fantasy, family	US	8.1	389774.0	7.6
1	Citizen Kane	MOVIE	1941	119	drama	US	8.3	433804.0	8.0
2	Casablanca	MOVIE	1942	102	drama, romance, war	US	8.5	558849.0	8.2
3	The Big Sleep	MOVIE	1946	116	thriller, crime	US	7.9	84494.0	7.7
4	The Maltese Falcon	MOVIE	1941	100	thriller, romance, crime	US	8.0	156603.0	7.8
...	...	...	...	...	...	...	...	...	...
3257	Covid Diaries NYC	MOVIE	2021	40	documentation	US	3.7	184.0	5.9
3275	Breathless	MOVIE	2021	106	crime, drama, thriller	DO	6.3	27.0	5.9
3279	Furry Friends Forever: Elmo Gets a Puppy	MOVIE	2021	26	animation	US	6.8	14.0	10.0
3283	Marlon Wayans: You Know What It Is	MOVIE	2021	58	comedy	US	3.8	224.0	5.4
3290	Algo Azul	MOVIE	2021	90	comedy	PA	5.9	50.0	2.0

Os dois datasets que iremos trabalhar precisam ser concatenados em um único dataset.

```
# Juntar as 2 bases e um único Dataset
df1= [dfNetflix,dfAmazon]
dataSet=pd.concat(df1)
```

Depois de concatenado, assim como o dataset da HBO, os dados precisam ser tratados para remover os caracteres especiais e as linhas que contém algum dado vazio.

```
# Remover linhas com indices duplicados
dataSet = dataSet[~dataSet.index.duplicated()]
```

```
# Iremos remover as linhas cujo o pais seja vazio
dataSet = dataSet[dataSet.production_countries !='']
```

```
# Remover aspas simples das colunas genres e production_countries
dataSet["genres"] = dataSet["genres"].str.replace("'", "")
dataSet["production_countries"] = dataSet["production_countries"].str.replace("'", "")
```

```
# Remover caracter -> [ ]
dataSet["genres"] = dataSet["genres"].str.replace("]","", regex=True)
dataSet["genres"] = dataSet["genres"].str.replace("[","", regex=True)
dataSet["production_countries"] = dataSet["production_countries"].str.replace("]","", regex=True)
dataSet["production_countries"] = dataSet["production_countries"].str.replace("[","", regex=True)
```

```
dataSet = dataSet.dropna()
```

```
# Ordenar os valores da coluna genres
dataSet["genres"] = [','.join(sorted(x.split(','))) for x in dataSet['genres']]
```

Foi preciso corrigir alguns valores, pois algumas linhas no campo genres, trazia a informação de “european”, informando que eram do gênero comedia europeia, romance europeu, essa distinção não será mantida.

Em alguns casos no campo production_countries no lugar de informar a sigla do país, estava trazendo o nome, isso também teve que ser tratado.

```
# Remover a parte de European dos generos
dataSet["genres"] = dataSet["genres"].str.replace("european",'')
```

```
# Converter o nome do Pais para a sigla
dataSet['production_countries'] = dataSet['production_countries'].replace("['Lebanon']",'LB')
dataSet['production_countries'] = dataSet['production_countries'].replace("['United States of America']",'US')
```

```
# Ordenar o dataset pelo ano de lançamento
dataSet = dataSet.sort_values(by='release_year')
```

```
# Eliminar as linhas que não tem pais
dataSet = dataSet[dataSet.production_countries !='']
```

Foi realizada a verificação se ainda existia algum valor vazio no dataset.

```
# Verificar se tem algum null nos dados
dataSet.isnull().sum()
```

```
title                 0
type                  0
release_year          0
runtime               0
genres                0
production_countries  0
imdb_score            0
imdb_votes            0
tmdb_score            0
streaming             0
dtype: int64
```

4. Análise e Exploração dos Dados

Os dados do dataset foram classificados entre “Bom” e “Ruim” com base em suas pontuações tanto no IMDB com TMDB, quantidade de execuções e quantidade de votos do IMDB.

Para isso foi usado o código abaixo:

```
# Classificação nos gostos do usuário BRENNO
#imdb >= 7 a pontuação tmdb >=6 o número de runtime maior que 70 e quantidade de votos deve ser maior que 400
filtro = [
    (dataSet['imdb_score'] >= 7) & (dataSet['tmdb_score'] >= 6) & (dataSet['runtime'] >= 70) & (dataSet['imdb_votes'] >= 400)
]

#Resultados
resultado = ['Bom']

#Novo coluna com o status de bom e 0
dataSet['Status'] = np.select(filtro, resultado)

#Tudo que for 0 substitui para Ruim
dataSet['Status'] = dataSet['Status'].str.replace("0",'Ruim')

#Verificar se todos os dados estão divididos entre "BOM" e "RUIM"
dataSet['Status'].unique()
```

```
array(['Ruim', 'Bom'], dtype=object)
```

Foi verificado que todos os valores da nova coluna chamada “Status” estão preenchidos apenas com valores “Ruim” e “Bom”.

Foi analisado a quantidade de shows, levando em consideração o ano de lançamento e o tipo.

Podemos ver pela imagem abaixo que a maioria dos dados que temos são de anos recentes de 2020 em diante.

```
# Quantidade de shows e movie produzidos no ano, por tipo
dfTiposShow=dataSet.groupby(['release_year','type']).size().reset_index(name='Total')
dfTiposShow
```

	release_year	type	Total
0	1914	MOVIE	1
1	1927	MOVIE	1
2	1929	MOVIE	1
3	1930	MOVIE	1
4	1931	MOVIE	1
...	...	...	...
131	2020	SHOW	299
132	2021	MOVIE	659
133	2021	SHOW	329
134	2022	MOVIE	184
135	2022	SHOW	176

Fizemos a verificação da quantidade produzida por ano e por tipo de streaming, para sabermos qual streaming teria mais peso nas análises futuras, podemos ver que a Netflix como era esperado tem quase ou mais que o dobro de programas.

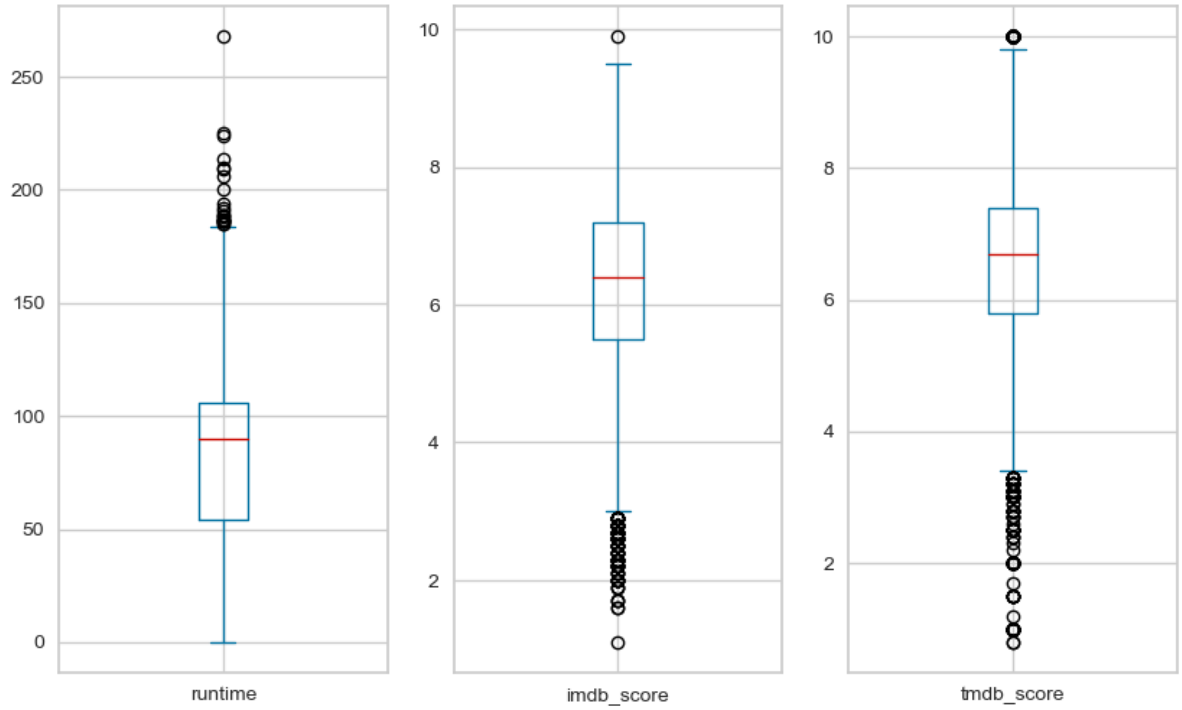
```
# Quantidade de programas produzidos no ano por streaming
dfStreaming=dataSet.groupby(['release_year','streaming']).size().reset_index(name='Total')
dfStreaming
```

	release_year	streaming	Total
0	1914	Amazon	1
1	1927	Amazon	1
2	1929	Amazon	1
3	1930	Amazon	1
4	1931	Amazon	1
...	...	...	...
144	2020	Netflix	621
145	2021	Amazon	345
146	2021	Netflix	643
147	2022	Amazon	55
148	2022	Netflix	305

Foi gerado o gráfico de boxplot para verificarmos como os dados se comportam e por possíveis outliers. Foi usado 2 bibliotecas diferentes que geram gráficos de boxplot, que são matplotlib e o plotly.

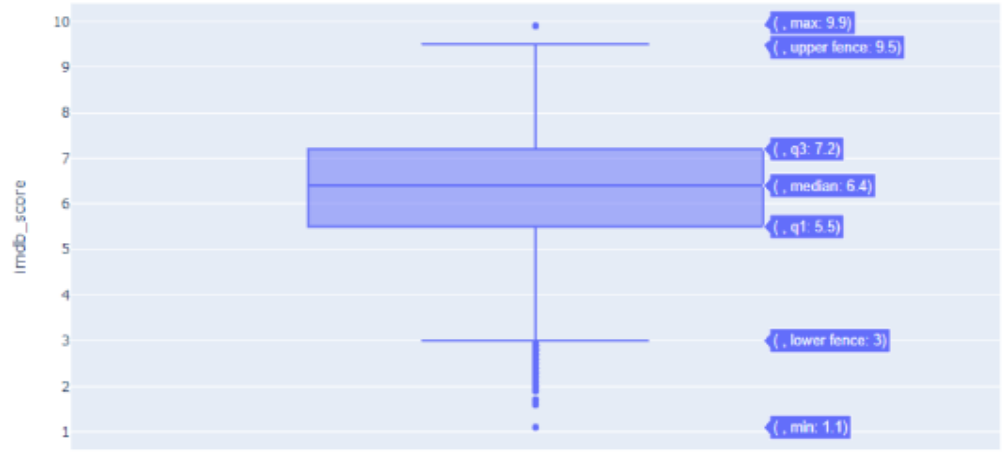
```
# Gráfico de plot analisando por ano de lançamento, acessos e pontuação dos programas
dfBox=dataSet[["streaming","runtime","imdb_score","tmdb_score"]]
dfBox.plot(kind="box",figsize=(10,6),subplots=True)
```

```
runtime          AxesSubplot(0.125,0.11;0.227941x0.77)
imdb_score    AxesSubplot(0.398529,0.11;0.227941x0.77)
tmdb_score    AxesSubplot(0.672059,0.11;0.227941x0.77)
dtype: object
```



Pelo gráfico resultante, podemos ver que temos vários outliers em nossos dados, iremos usar a biblioteca ploty para podermos visualizar cada um deles separados e de uma forma melhorada.

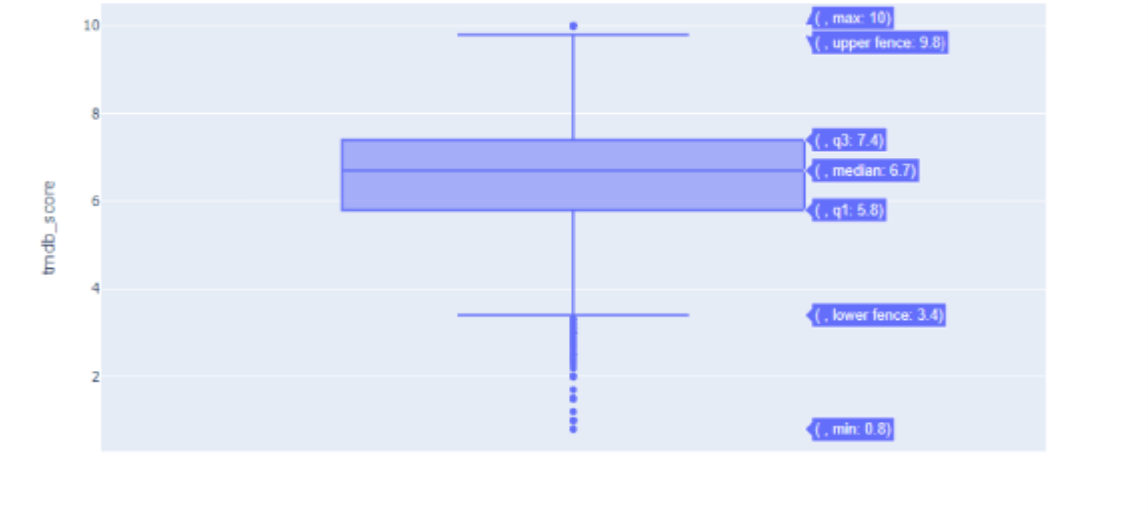
```
In [36]: # Gráfico de plot melhorado, sobre a pontuação do imdb
grafico = px.box(dataSet, y = "imdb_score")
grafico.show()
```



Podemos ver pelo gráfico acima, referente à pontuação do IMDB, que os programas com pontuação acima de 9.5 e abaixo de 3, estão se comportando como outliers nesse caso.

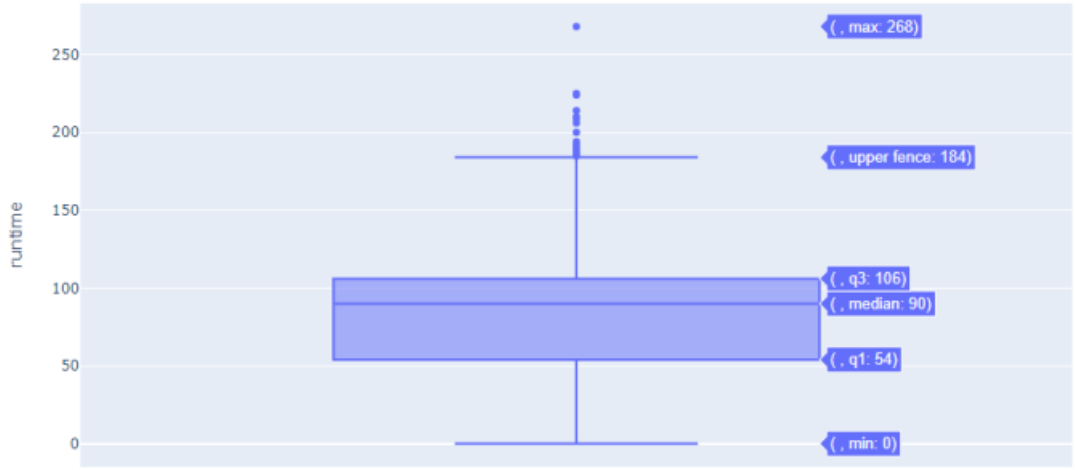
Na figura abaixo, podemos verificar que as pontuações acima de 9.8 e abaixo de 3.4 são os nosso outliers.

```
# Gráfico de plot melhorado, sobre a pontuação do tmdb
grafico = px.box(dataSet, y = "tmdb_score")
grafico.show()
```



No gráfico abaixo foi avaliado os possíveis outliers das execuções:

```
In [38]: # Gráfico de plot melhorado, sobre os acessos
grafico = px.box(dataSet, y = "runtime")
grafico.show()
```



Depois foi impresso esses dados na tela para verificação.

Começando pelos dados de “runtime”, seguidos das pontuações da plataforma TMDB e depois da IMDB.

RUNTIME

```
#Verificando os possivéis outliers - acessos
runtime_maior =dataSet[dataSet['runtime'] > 184]
runtime_maior = runtime_maior.reset_index()
runtime_maior[["title","streaming","release_year","imdb_score","tmdb_score"]]
```

	title	streaming	release_year	imdb_score	tmdb_score
0	Saladin the Victorious	Netflix	1963	7.6	7.100
1	Wyatt Earp	Netflix	1994	6.7	6.751
2	Hum Aapke Hain Koun..!	Netflix	1994	7.5	6.500
3	Titanic	Netflix	1997	7.9	7.878
4	Kuch Kuch Hota Hai	Netflix	1998	7.6	7.700
5	Kabhi Khushi Kabhie Gham	Netflix	2001	7.4	7.900
6	Lagaan: Once Upon a Time in India	Netflix	2001	8.1	7.300
7	Dil Hai Tumhaara	Amazon	2002	5.4	6.000
8	Kal Ho Naa Ho	Netflix	2003	7.9	7.500
9	Swades	Netflix	2004	8.1	7.400
10	Lakshya	Netflix	2004	7.8	6.600
11	A Lion in the House	Netflix	2006	8.7	6.500
12	Sivaji: The Boss	Netflix	2007	7.5	7.100
13	Jodhaa Akbar	Netflix	2008	7.5	7.400
14	What's Your Raashee?	Netflix	2009	4.8	4.900
15	The Hateful Eight	Netflix	2015	7.8	7.737
16	Arjun Reddy	Amazon	2017	8.2	7.200
17	Jab Harry Met Sejal	Netflix	2017	5.0	6.500
18	Avane Srimannarayana	Amazon	2019	7.9	7.500
19	The Irishman	Netflix	2019	7.8	7.600
20	On the Trail of UFOs	Amazon	2020	5.8	4.000
21	RRR	Netflix	2022	8.0	7.800

TMDB

```
#Verificando os possivéis outliers - tmdb
piores_notas_tmdb=dataSet[dataSet['tmdb_score'] < 3.4]
piores_notas_tmdb=piores_notas_tmdb.reset_index()
piores_notas_tmdb[["title","streaming","release_year","imdb_score","tmdb_score"]]
```

	title	streaming	release_year	imdb_score	tmdb_score
0	The Luck of Roaring Camp	Amazon	1937	5.9	3.0
1	Sky Racket	Amazon	1937	4.8	2.0
2	Two Weeks to Live	Amazon	1943	5.6	2.0
3	The Slime People	Amazon	1963	2.7	3.2
4	The Next One	Amazon	1984	4.7	1.5
...	...	...	...	...	...
180	Johnny Test's Ultimate Meatloaf Quest	Netflix	2021	7.1	2.5
181	Mommy Issues	Netflix	2021	5.4	1.0
182	Let's Tie the Knot, Honey!	Netflix	2022	4.4	2.0
183	Sumaira Shaikh: Dongri Danger	Amazon	2022	4.9	1.0
184	Queer Eye Germany	Netflix	2022	7.2	1.0

185 rows × 5 columns

```
#Verificando os possivéis outliers - tmdb
maiores_notas_tmdb=dataSet[dataSet['tmdb_score'] > 9.8]
maiores_notas_tmdb=maiores_notas_tmdb.reset_index()
maiores_notas_tmdb[["title","streaming","release_year","imdb_score","tmdb_score"]]
```

	title	streaming	release_year	imdb_score	tmdb_score
0	Brothers O'Toole	Amazon	1973	4.5	10.0
1	Pink Zone	Netflix	2007	5.8	10.0
2	Primal Grill with Steven Raichlen	Amazon	2009	8.0	10.0
3	Alonzo Bodden: Who's Paying Attention	Amazon	2011	7.1	10.0
4	Little Baby Bum	Amazon	2011	6.4	10.0
...	...	...	...	...	...
99	Super PupZ	Netflix	2022	6.6	10.0
100	Lov3	Amazon	2022	5.9	10.0
101	The Big Shot Game Show	Netflix	2022	6.2	10.0
102	Senzo: Murder of a Soccer Star	Netflix	2022	5.2	10.0
103	Dogtown 2	Amazon	2022	3.2	10.0

104 rows × 5 columns

IMDB

```
#Verificando os possivéis outliers - imdb
piores_notas_imdb=dataSet[dataSet['imdb_score'] < 3]
piores_notas_imdb=piores_notas_imdb.reset_index()
piores_notas_imdb[["title","streaming","release_year","imdb_score","tmdb_score"]]
```

	title	streaming	release_year	imdb_score	tmdb_score
0	The Slime People	Amazon	1963	2.7	3.2
1	Dostana	Netflix	1980	2.1	4.9
2	Mutants	Amazon	2008	2.3	3.1
3	Attack of the Giant Leeches	Amazon	2008	1.9	3.0
4	Agyaat	Netflix	2009	2.9	4.1
...	...	...	...	...	...
99	365 Days: This Day	Netflix	2022	2.5	5.8
100	Rodrigo Sant'anna: I'm Here, I'm Queer!	Netflix	2022	2.6	7.5
101	Byron Baes	Netflix	2022	2.6	6.0
102	Hype House	Netflix	2022	2.1	5.8
103	He's Expecting	Netflix	2022	2.0	4.0

104 rows × 5 columns

```
#Verificando os possivéis outliers - imdb
maiores_notas_imdb=dataSet[dataSet['imdb_score'] > 9.5]
maiores_notas_imdb=maiores_notas_imdb.reset_index()
maiores_notas_imdb[["title","streaming","release_year","imdb_score","tmdb_score"]]
```

Foi verificado a tabulação cruzada do dataset com os campos type e streaming, para comparar a dimensão de dados por streaming.

```
# Analisando a tabulação cruzada entre os campos do Dataset (type - streaming)
pd.crosstab(dataSet["type"], dataSet["streaming"])
```

streaming	Amazon	Netflix
type		
MOVIE	2566	3196
SHOW	442	1823

Foi verificado pelo gráfico de barras que os dados da Netflix eram de um pouco mais de 2 mil à mais que o da Amazon. Os dados da Netflix representam 62.5% das amostrar, conforme o outro gráfico de pizza.

```
# Gráfico de barra, agrupando por streaming
dataSet['streaming'].value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```

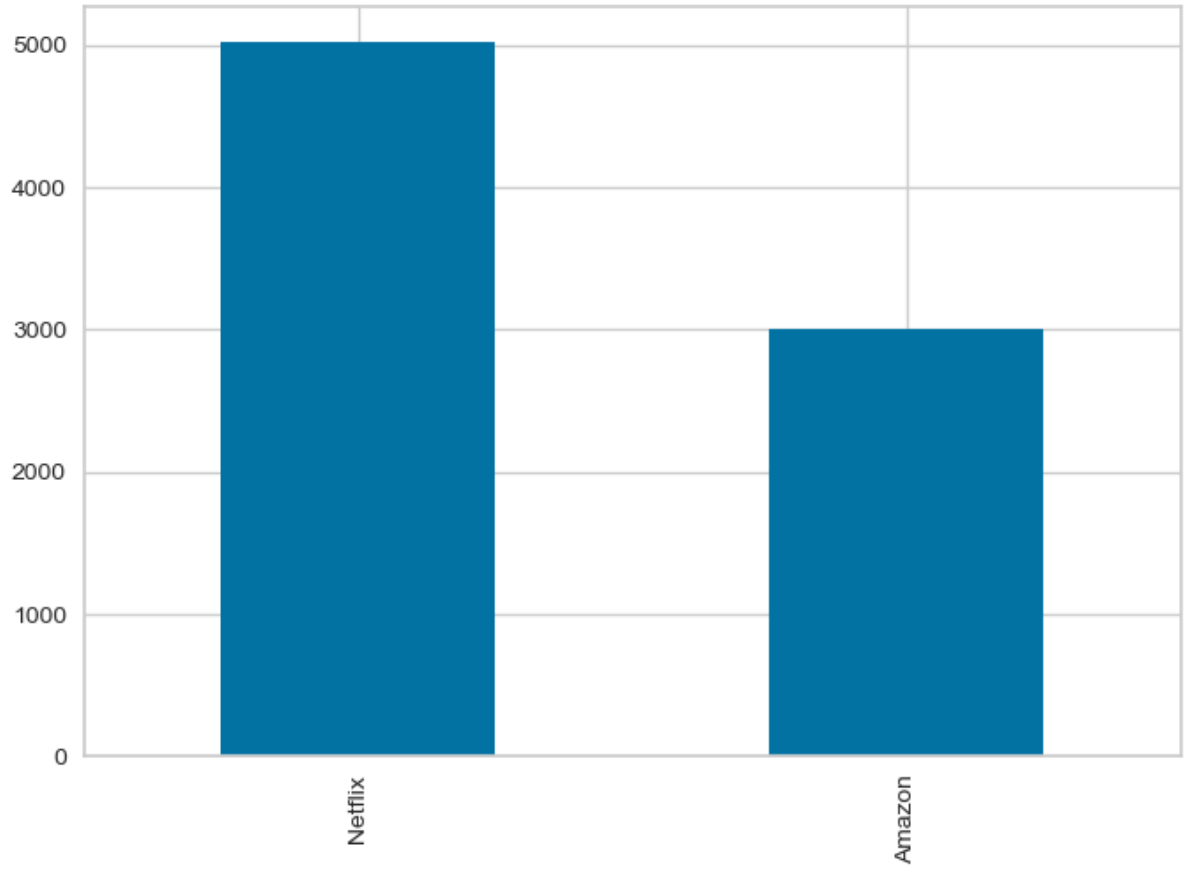
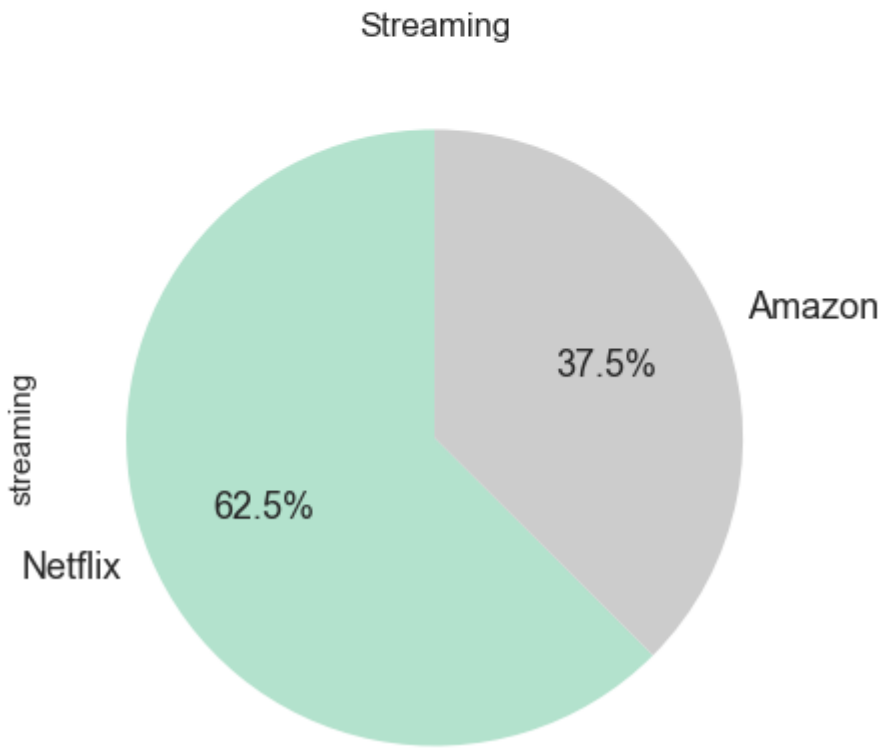


Gráfico de Pizza separado por Streaming

```
kwargs = dict(
    startangle = 90,
    colormap = 'Pastel2',
    fontsize = 13,
    explode = None,
    figsize=(60,5),
    autopct = '%1.1f%%',
    title = 'Streaming'
 )
```

```
dataSet['streaming'].value_counts().plot.pie(**kwargs)
```

```
<AxesSubplot:title={'center':'Streaming'}, ylabel='streaming'>
```



Após essa etapa foi verificado qual tipo de programa tinha mais peso nos dados que estamos analisando.

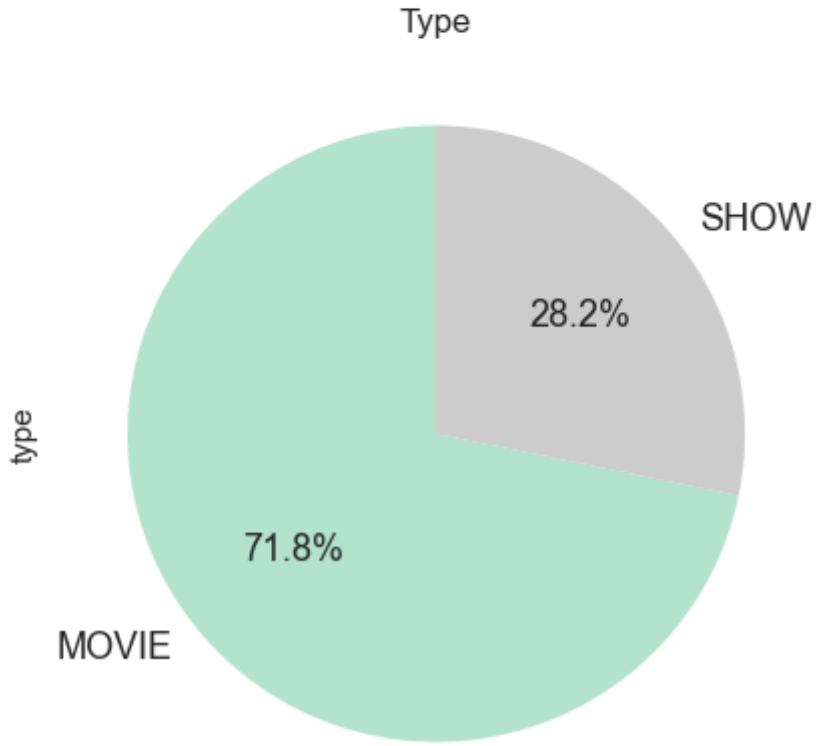
Foi usado o gráfico de pizza abaixo.

Gráfico de Pizza separado por tipo

```
kwargs = dict(
    startangle = 90,
    colormap = 'Pastel2',
    fontsize = 13,
    explode = None,
    figsize=(60,5),
    autopct = '%1.1f%%',
    title = 'Type'
 )
```

```
dataSet['type'].value_counts().plot.pie(**kwargs)
```

```
<AxesSubplot:title={'center':'Type'}, ylabel='type'>
```



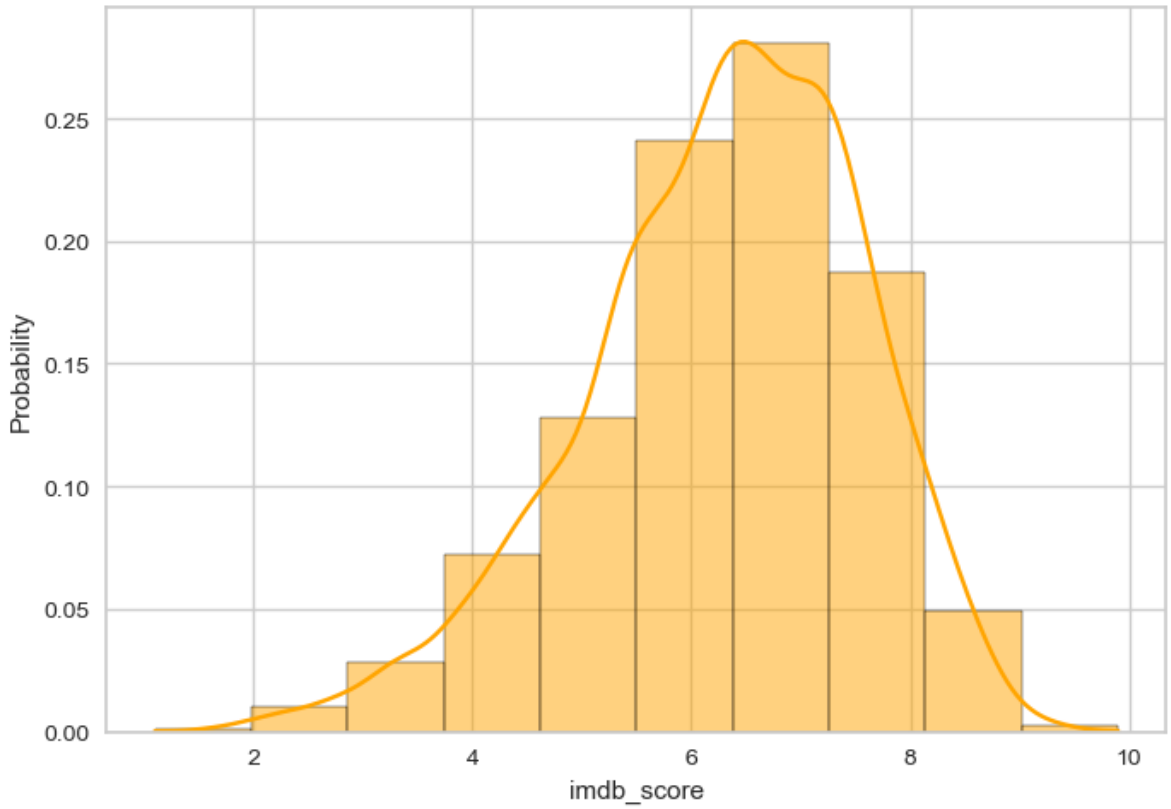
5. Criação de Modelos de Machine Learning

Foi utilizado o modelo de Regressão Linear para poder identificar uma relação entre as 2 grandezas usadas como pontuação dos programas apresentados pelos streamings analisados, essas grandezas era a pontuação do IMDB que é a principal e mais famosa e a TMDB como uma segunda avaliação, o objetivo era verificar se elas se correlacionam em uma distribuição normal.

Foi criado um gráfico de histograma para verificar como as duas grandezas se comportam.

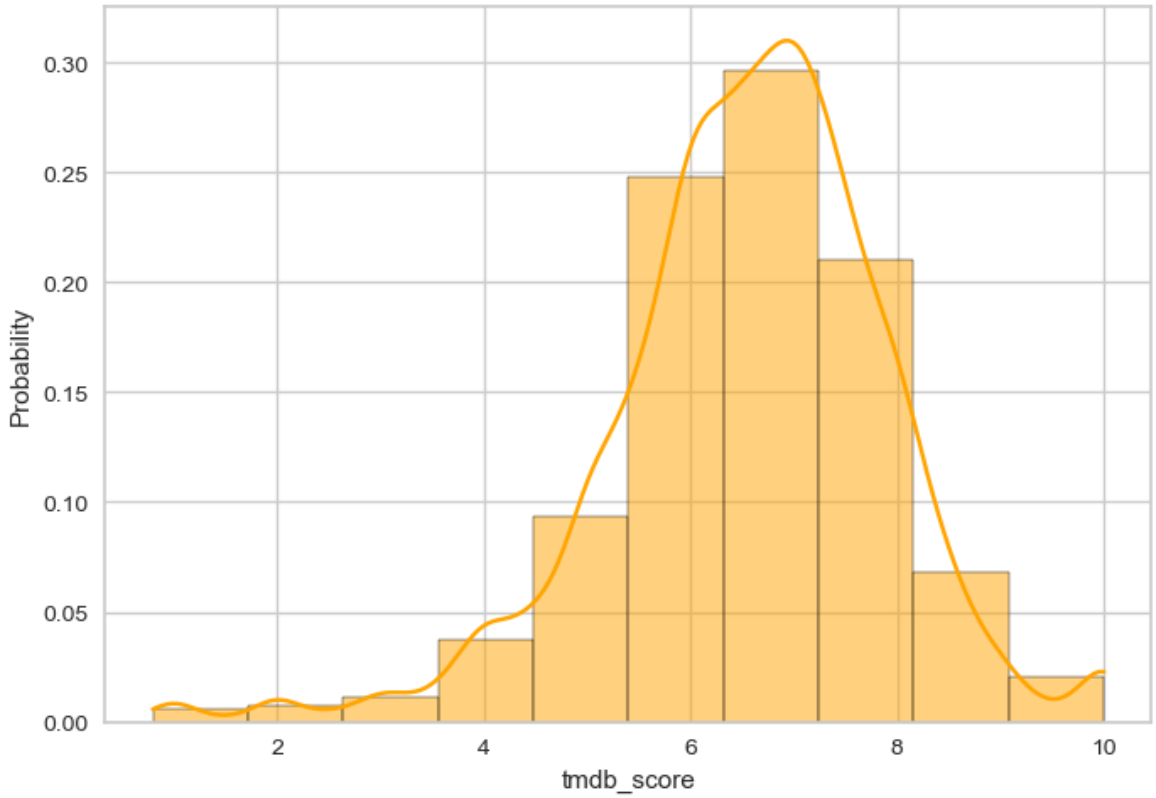
```
# Gráfico de histograma da pontuação - IMDB
sns.histplot(dataSet,x= "imdb_score", bins=10, color="orange", kde=True,stat="probability")
```

```
<AxesSubplot:xlabel='imdb_score', ylabel='Probability'>
```



```
# Gráfico de histograma da pontuação - TMDB
sns.histplot(dataSet,x= "tmdb_score", bins=10, color="orange", kde=True,stat="probability")
```

```
<AxesSubplot:xlabel='tmdb_score', ylabel='Probability'>
```



5. Criação de Modelos de Machine Learning

Regressão Linear

Foi criado um gráfico para verificar como os dados estão dispersos por país, mas por questão de curiosidade mesmo. Foi verificado um padrão curioso de concentração de shows em determinados países e filmes em outros, algo que eu não esperava. O gráfico abaixo foi gerado, pois várias tentativas mostravam que a regressão linear, não era viável, o gráfico abaixo foi mais uma tentativa de achar um motivo para isso. Será explicado no próximo tópico.

A regressão linear foi usada para prever o valor de uma variável com base no valor de outra variável, foi usado a pontuação do IMDB para poder prever a pontuação do TMDB.

As vantagens:

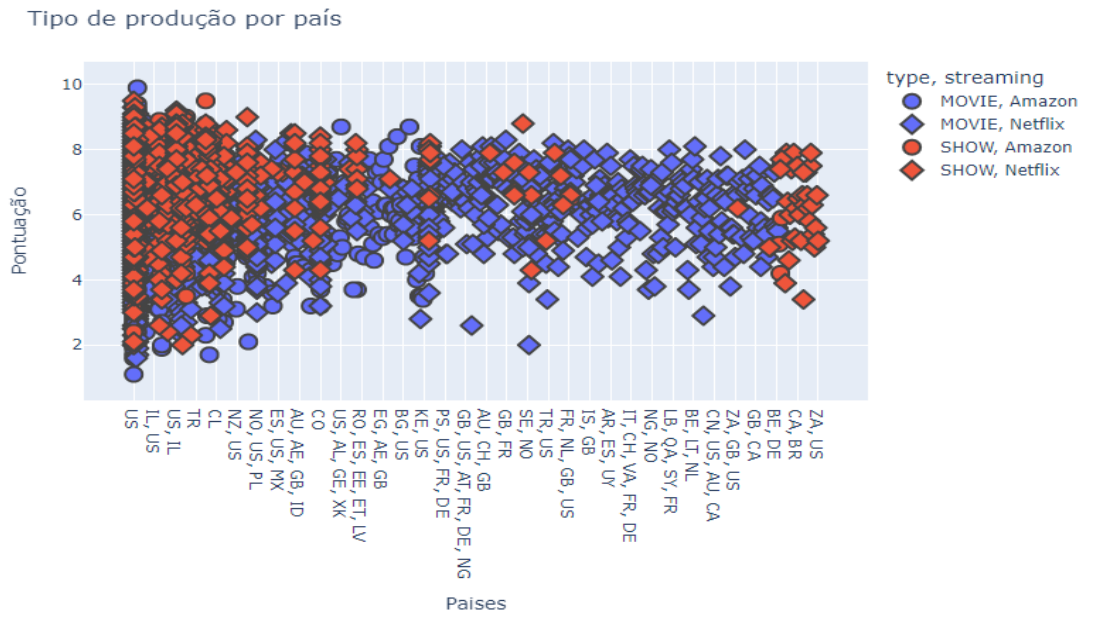
- Simples de implementar
- Prever o futuro
- Possível realizar ajustes ou pode se aplicar técnicas de dimensionalidade, técnicas de regularização e validação cruzada

As desvantagens:

- Os outliers podem impactar muito a previsão
- Limitado a relacionamentos lineares
- Os dados devem ser independentes

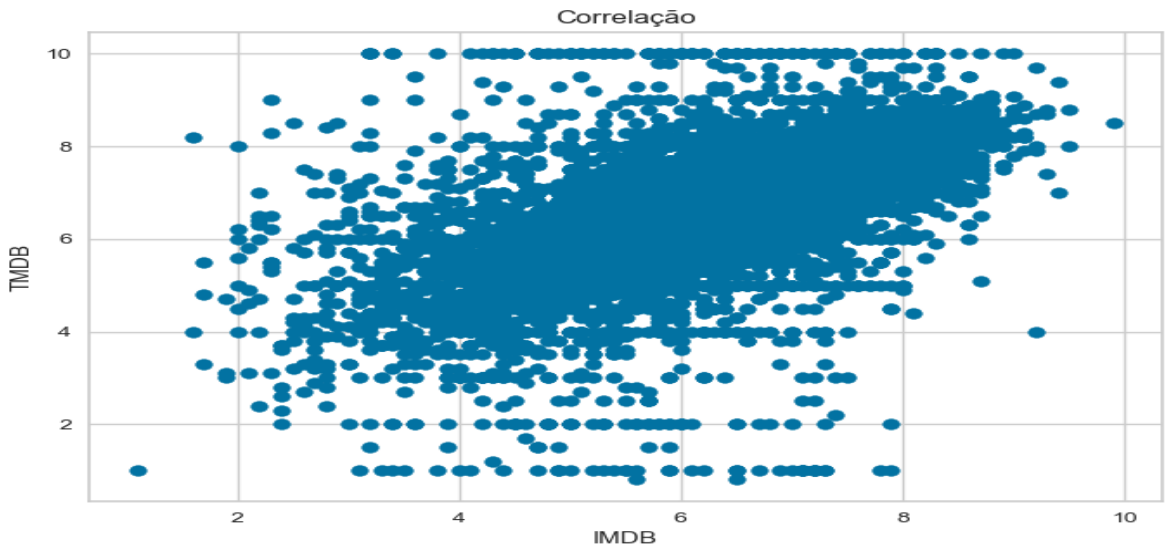
TMDB é a nossa variável dependente e a IMDB a nossa variável independente.

```
# Gráfico de dispersão com base nos paises, pontuação do imdb e o tipo.
fig = px.scatter(dataSet, x = "production_countries",y = "imdb_score" , symbol = 'streaming', color = "type",
                 hover_name = "type", log_x = False, width = 800)
fig.update_traces(marker=dict(size = 12,line = dict(width = 2)),selector=dict(mode = 'markers'))
fig.update_layout(title = 'Tipo de produção por país')
fig.update_xaxes(title = 'Paises')
fig.update_yaxes(title = 'Pontuação')
fig.show()
```



Foi verificado a correlação das duas grandezas para tentar identificar um padrão entre elas.

```
# Correlação entre as pontuações de IMDB e TMDB
plt.scatter(dataSet.imdb_score,dataSet.tmdb_score)
plt.title("Correlação")
plt.xlabel("IMDB")
plt.ylabel("TMDB")
plt.grid(True)
plt.show()
```



Foi criado outro dataset para poder analisar melhor a relação entre as grandezas existentes, envolvendo o runtime, imdb_score, tmdb_score e imdb_votes.

Esse novo dataset foi chamado de dfRegressão.

```
# Criar um Dataframe com apenas os campos nessa análise
dfRegressao = dataSet[["streaming","runtime","imdb_score","imdb_votes","tmdb_score"]]
dfRegressao
```

	streaming	runtime	imdb_score	imdb_votes	tmdb_score
180	Amazon	63	5.8	251.0	4.4
554	Amazon	78	6.3	810.0	7.2
633	Amazon	66	5.4	412.0	4.2
549	Amazon	74	6.1	532.0	5.7
469	Amazon	82	6.1	478.0	5.2
...	...	...	...	...	...
5598	Netflix	179	6.0	2248.0	6.7
5066	Netflix	100	6.2	3434.0	6.5
5067	Netflix	167	7.3	2563.0	6.2
9058	Amazon	138	7.8	645.0	6.3
4708	Netflix	83	4.6	4583.0	5.9

Foi utilizado a tabela de correlação, pega nos materias de estudos utilizados para formar a regressão linear. Podemos que correlação entre as grandezas IMDB e TMDB (pontuação) é moderada.

Coeficiente de correlação (r)	Correlação Positiva	Coeficiente de correlação (r)	Correlação Negativa
r = 1	Perfeita	r = - 1	Perfeita
0,95 ≤ r < 1	Muito forte	- 0,95 ≤ r < -1	Muito forte
0,8 ≤ r < 0,95	Forte	-0,8 ≤ r < -0,95	Forte
0,5 ≤ r < 0,8	Moderada	-0,5 ≤ r < -0,8	Moderada
0 ≤ r < 0,5	Fraca	0 ≤ r < -0,5	Fraca

```
# Avaliar as correlações das grandezas
dataSet.corr()
```

	release_year	runtime	imdb_score	imdb_votes	tmdb_score
release_year	1.000000	-0.049670	-0.019386	-0.100526	0.095120
runtime	-0.049670	1.000000	-0.160509	0.104245	-0.257727
imdb_score	-0.019386	-0.160509	1.000000	0.188369	0.568936
imdb_votes	-0.100526	0.104245	0.188369	1.000000	0.122148
tmdb_score	0.095120	-0.257727	0.568936	0.122148	1.000000

Diante do resultado de correlação moderada eu tive a necessidade de verificar a correlação entre as 2 grandezas por streaming para verificar se alguns dos streamings era o responsável por jogar a correlação para baixo. Aparentemente não, o resultado por streaming mostra uma correlação bem próximas.

```
# Correlação das pontuações (Amazon)
dfNet = dataSet.where(dataSet["streaming"]=="Amazon")
dfNet = dfNet.dropna()
dfNet[["imdb_score","tmdb_score"]].corr()
```

	imdb_score	tmdb_score
imdb_score	1.000000	0.502499
tmdb_score	0.502499	1.000000

```
# Correlação das pontuações (Netflix)
dfNet = dataSet.where(dataSet["streaming"]=="Netflix")
dfNet = dfNet.dropna()
dfNet[["imdb_score","tmdb_score"]].corr()
```

	imdb_score	tmdb_score
imdb_score	1.000000	0.580052
tmdb_score	0.580052	1.000000

Foi aplicada o algoritmo de regressão para podermos verificar as outras variáveis como por exemplo a raiz quadrada que indicaria a relação entre as duas grandezas alvos. Como podemos ver pelas imagens abaixo a raiz quadrada ajustada ficou em 0.3 que é muito baixo.

```
# Regressão usando as grandezas runtime e imdb_score
regressao = smf.ols("imdb_score ~ tmdb_score", data =dfRegressao).fit()
print(regressao.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.324
Model:                            OLS   Adj. R-squared:                  0.324
Method:                 Least Squares   F-statistic:                     3841.
Date:                Tue, 31 Jan 2023   Prob (F-statistic):               0.00
Time:                        09:08:42   Log-Likelihood:                -11807.
No. Observations:                8027   AIC:                         2.362e+04
Df Residuals:                    8025   BIC:                         2.363e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.7694      0.058     47.817      0.000       2.656       2.883
tmdb_score     0.5364      0.009     61.974      0.000       0.519       0.553
==============================================================================
Omnibus:                      880.065   Durbin-Watson:                   1.903
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1772.540
Skew:                          -0.703   Prob(JB):                         0.00
Kurtosis:                       4.823   Cond. No.                         33.7
==============================================================================
```

Foi calculado o coeficiente, tanto linear como o Angular

```
# Os Coeficientes (Linear/ Angular)
coefs = pd.DataFrame(regressao.params)
coefs.columns = ["Os_Coeficientes"]
print(coefs)
```

```
            Os_Coeficientes
Intercept          2.769399
tmdb_score         0.536386
```

Foi gerado depois as previsões e os resíduos que mostra a distância que o valor está da reta ajustada (reta de regressão).

```
# Previssão para cada um dos valores do imdb com base no valor do tmdb
regressao.predict()
```

```
array([5.1294966 , 6.63137705, 5.02221943, ..., 6.09499118, 6.14862976,
       5.93407541])
```

```
dfRegressao.head()
```

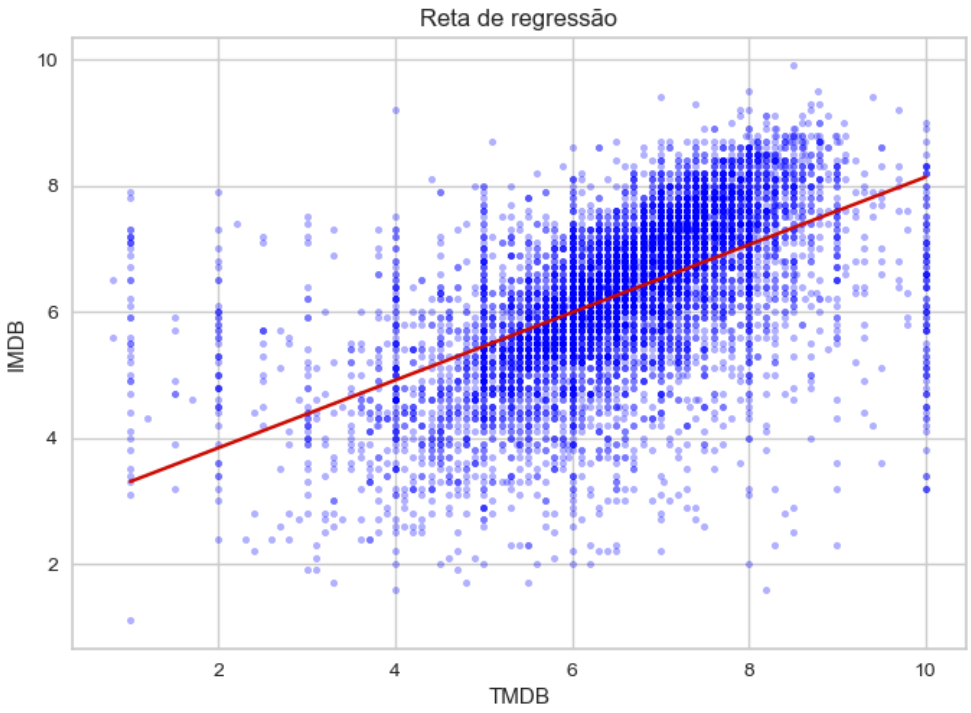
	streaming	runtime	imdb_score	imdb_votes	tmdb_score
180	Amazon	63	5.8	251.0	4.4
554	Amazon	78	6.3	810.0	7.2
633	Amazon	66	5.4	412.0	4.2
549	Amazon	74	6.1	532.0	5.7
469	Amazon	82	6.1	478.0	5.2

```
# Residuo é distancia entre os dados e a reta ajustada. Mostra a distancia que o valor está da reta ajustada (reta de regressão)
residuos = regressao.resid
residuos
```

```
180     0.670503
554    -0.331377
633     0.377781
549     0.273202
469     0.541395
          ...
5598   -0.363184
5066   -0.055907
5067    1.205009
9058    1.651370
4708   -1.334075
Length: 8027, dtype: float64
```

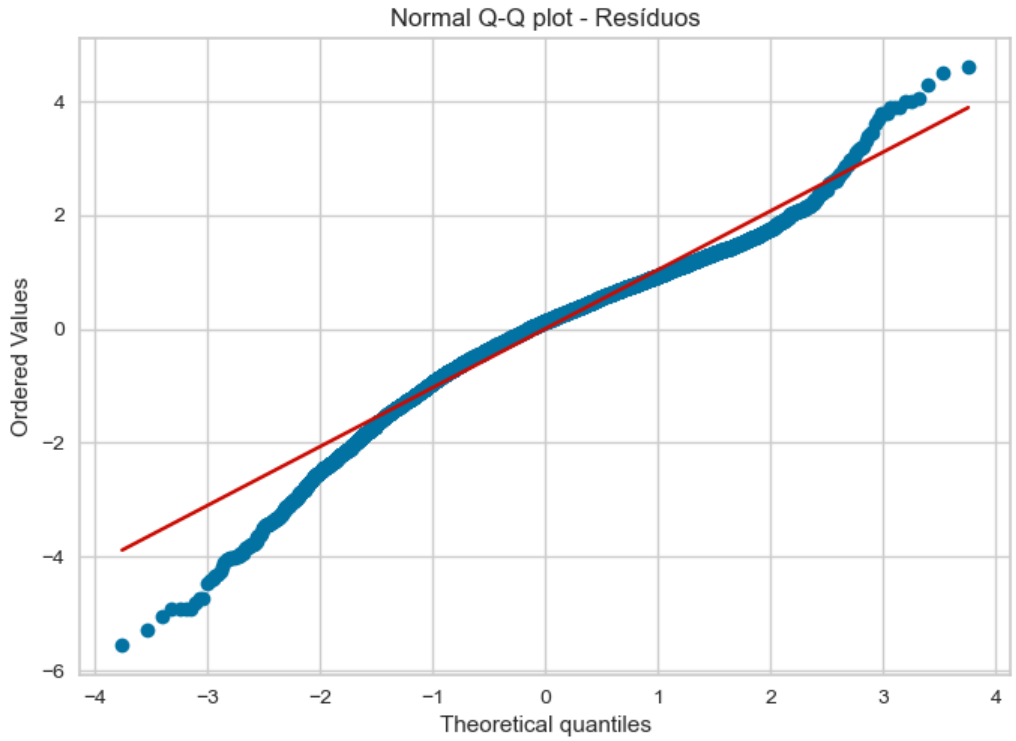
Foi gerado a reta de regressão, para verificar se os dados estão em uma distribuição normal. Podemos verificar que existe uma divergência alta nas extremidades.

```
# Gerar o gráfico da Reta de Regressão
plt.scatter(y=dfRegressao.imdb_score,x=dfRegressao.tmdb_score, color='blue', s=10, alpha=.3)
#X_plot = np.linspace(min(dfRegressao.tmdb_score), max(dfRegressao.tmdb_score), len(dfRegressao.tmdb_score))
X_plot = np.linspace(1,10)
plt.plot(X_plot, X_plot*regressao.params[1] + regressao.params[0], color='r')
plt.title('Reta de regressão')
plt.ylabel('IMDB')
plt.xlabel('TMDB')
plt.show()
```



Com isso foi preciso gerar o gráfico abaixo para verificar se os dados estão dispostos em uma distribuição normal. Como verificado acima nas extremidades a uma divergência, porém olhando o intervalo do eixo x que compreende de -1.5 até 2.8 mais ou menos os dados parecem indicar uma distribuição normal, consegue realizar uma boa previsão com uma taxa de acerto boa, no meio, mas erra muito nas extremidades.

```
# É uma Distribuição normal - para ser distribuição normal eles precisam concindir com a linha vermelha.
# Consegue realizar uma previsão boa nos valores do meio da reta, ams erra muito nas extremidades
stats.probplot(residuos, dist="norm", plot=plt)
plt.title("Normal Q-Q plot - Resíduos")
plt.show()
```



Foi preciso realizar a verificação estatística com as teses de Shapiro-Wilk, mas o volume de dados era muito grande, embora mesmo reduzindo o volume dos dados em outros testes o resultado também não foi dos melhores, foi usado o teste de Lilliefors e o de normalidade de Anderson.

O resultado esperado era p-valor igual ou superior a 0,05 ou 5%, porém em ambos os testes o resultado deu negativo para uma distribuição normal.

```
# Teste de Shapiro-Wilk
# Nível de significância de 0,05 ou 5%
# Quando p > 0,05(Distribuição normal)
# Teste de Normalidade dos resíduos - pvalue > 0.05 para ter uma distribuição normal (Para elementos menor que 5000)
stats.shapiro(residuos)
```

```
C:\PUC\ANACONDA\lib\site-packages\scipy\stats\_morestats.py:1800: UserWarning:

p-value may not be accurate for N > 5000.
```

```
ShapiroResult(statistic=0.9657446146011353, pvalue=6.8007396681072725e-40)
```

```
# Teste Lilliefors
statsmodels.stats.diagnostic.lilliefors(dataSet.imdb_score, dist="norm")
```

```
(0.05887773223296189, 0.0009999999999998899)
```

```
# Teste de normalidade de Anderson
ad_stat, ad_critico, ad_teorico = stats.anderson(residuos, 'norm')
print("O Valor da estatística calculada: " ,ad_stat)
print("Os valores : ",ad_critico)
print("Os níveis de significancia: ",ad_teorico)
```

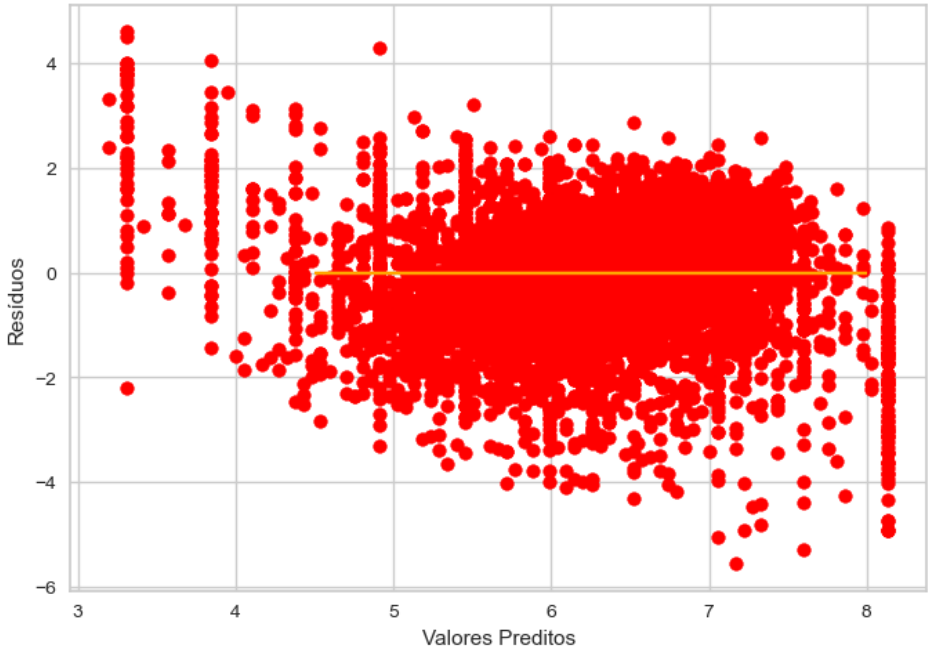
```
O Valor da estatística calculada:  66.51603506568972
Os valores :  [0.576 0.656 0.787 0.918 1.091]
Os níveis de significancia:  [15.  10.   5.   2.5  1. ]
```

```
if ad_stat < ad_critico[2]:
    print("Com " + str(100 - ad_teorico[2]) + "% de confiança, os dados são similares a uma distribuição normal"
          +"segundo o teste de AD")
else:
    print("Com " + str(100 - ad_teorico[2]) + "% de confiança, os dados não são similares a uma distribuição normal"
          +"segundo o teste de AD")
```

```
Com 95.0% de confiança, os dados não são similares a uma distribuição normalsegundo o teste de AD
```

Foi realizado também a análise da homoscedasticidade dos resíduos, onde foi verificado que os resíduos não são homogêneos para todos os valores de Y.

```
# Análise da Homocedasticidade dos resíduos
# Variação constante dos resíduos, além da distruição normal
# Tem que gerar algo parecido com um retangulo
#Ao analisar o gráfico acima, é possível perceber que os resíduos não são homogêneos para todos os valores de Y
# São mais homogeneos nos intervalos de x entre 5 e 7,5. Mais ou menos
plt.scatter(y=residuos, x=regressao.predict(), color='red')
plt.hlines(y=0, xmin=4.5, xmax=8, color='orange')
plt.ylabel('Resíduos')
plt.xlabel('Valores Preditos')
plt.show()
```



Naive Bayes

Baseado no teorema de Bayes, gera uma tabela de probabilidades a partir de uma técnica de classificação de dados. De fácil implantação pode ser usado para previsões, com duas ou mais variáveis, filtragem de conteúdo, avaliação e recomendação.

Vantagens

- Fácil de implementar
- Rápido
- Se a suposição de independência se mantiver, ela funcionara com mais eficiência do que outros algoritmos
- Requer menos dados de treinamento
- É altamente escalável
- Pode fazer previsões probabilísticas

- Pode lidar com dados contínuos e discretos
- Insensível a características irrelevantes
- Pode funcionar facilmente com valores ausentes
- Fácil de atualizar na chegada de novos dados
- Mais adequado para problemas de classificação de texto.

Desvantagens

- A forte suposição de que os recursos são independentes, o que dificilmente se aplica as aplicações da vida real
- Escassez de dados
- Frequência zero, insto é, se a categoria de qualquer variável categórica não for vista no conjunto de dados de treinamento, o modelo atribuíra uma probabilidade zero a essa categoria e, portanto, uma previsão não poderá ser feita.

Para comer os estudos filtramos o dataset original com as colunas desejadas.

```
dataSet = dataSet[["title","type","release_year","runtime","genres","production_countries","imdb_score","imdb_votes",
                   "tmdb_score","Status"]]
dataSet
```

	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb_votes	tmdb_score	Status
180	A Florida Enchantment	MOVIE	1914	63	comedy,fantasy	US	5.8	251.0	4.4	Ruim
554	The Love of Sunya	MOVIE	1927	78	romance,drama	US	6.3	810.0	7.2	Ruim
633	The Racketeer	MOVIE	1929	66	crime, thriller,drama	US	5.4	412.0	4.2	Ruim
549	Danger Lights	MOVIE	1930	74	drama	US	6.1	532.0	5.7	Ruim
469	Behind Office Doors	MOVIE	1931	82	drama,romance	US	6.1	478.0	5.2	Ruim
...	...	...	...	...	...	...	...	...	...	...
5598	How I Fell in Love with a Gangster	MOVIE	2022	179	crime,drama	PL	6.0	2248.0	6.7	Ruim
5066	Silverton Siege	MOVIE	2022	100	crime, drama, thriller,action	ZA	6.2	3434.0	6.5	Ruim
5067	Jaadugar	MOVIE	2022	167	comedy, fantasy, romance, sport,drama	IN	7.3	2563.0	6.2	Bom
9058	Saani Kaayidham	MOVIE	2022	138	action, crime,drama	IN	7.8	645.0	6.3	Bom
4708	Umma	MOVIE	2022	83	drama,horror	US	4.6	4583.0	5.9	Ruim

8027 rows × 10 columns

Depois separamos as previsões da classificação (Status).

```
# Separar as previssões da classificação (Status)
previsores = dataSet.iloc[:,0:9].values
classe = dataSet.iloc[:,9].values
```

Precisamos transformar os atributos categóricos em atributos númericos, passando o índice de cada coluna categórica.

```
# Transformação dos atributos categóricos em atributos numéricos, passando o índice de cada coluna categórica
labelencoder1 = LabelEncoder()
previsores[:,0] = labelencoder1.fit_transform(previsores[:,0])

labelencoder2 = LabelEncoder()
previsores[:,1] = labelencoder1.fit_transform(previsores[:,1])

labelencoder3 = LabelEncoder()
previsores[:,4] = labelencoder1.fit_transform(previsores[:,4])

labelencoder4 = LabelEncoder()
previsores[:,5] = labelencoder1.fit_transform(previsores[:,5])
```

Logo em seguida realizamos o treinamento do modelo.

```
x_treinamento, x_teste,y_treinamento, y_teste = train_test_split(previsores,classe,test_size= 0.3, random_state=0)
x_teste
```

```
array([[6661, 0, 2018, ..., 6.1, 3220.0, 5.6],
       [4023, 1, 2021, ..., 8.4, 77885.0, 8.0],
       [1064, 0, 2009, ..., 6.7, 33051.0, 7.4],
       ...,
       [7157, 0, 2016, ..., 3.6, 88.0, 2.0],
       [1603, 0, 2020, ..., 4.2, 73228.0, 5.9],
       [2585, 1, 2000, ..., 7.2, 4528.0, 7.7]], dtype=object)
```

```
# Criação e treinamento do modelo
naive_bayes = GaussianNB()
naive_bayes.fit(x_treinamento, y_treinamento)
```

Logo em seguida criamos as previssões com os registros de testes criado acima e geramos a matriz de confusão para poder avaliar o desempenho do nosso modelo.

```
# Previssões utilizando os registros de teste
previsoes = naive_bayes.predict(x_teste)
previsoes
```

```
array(['Ruim', 'Ruim', 'Ruim', ..., 'Ruim', 'Ruim', 'Ruim'], dtype='<U4')
```

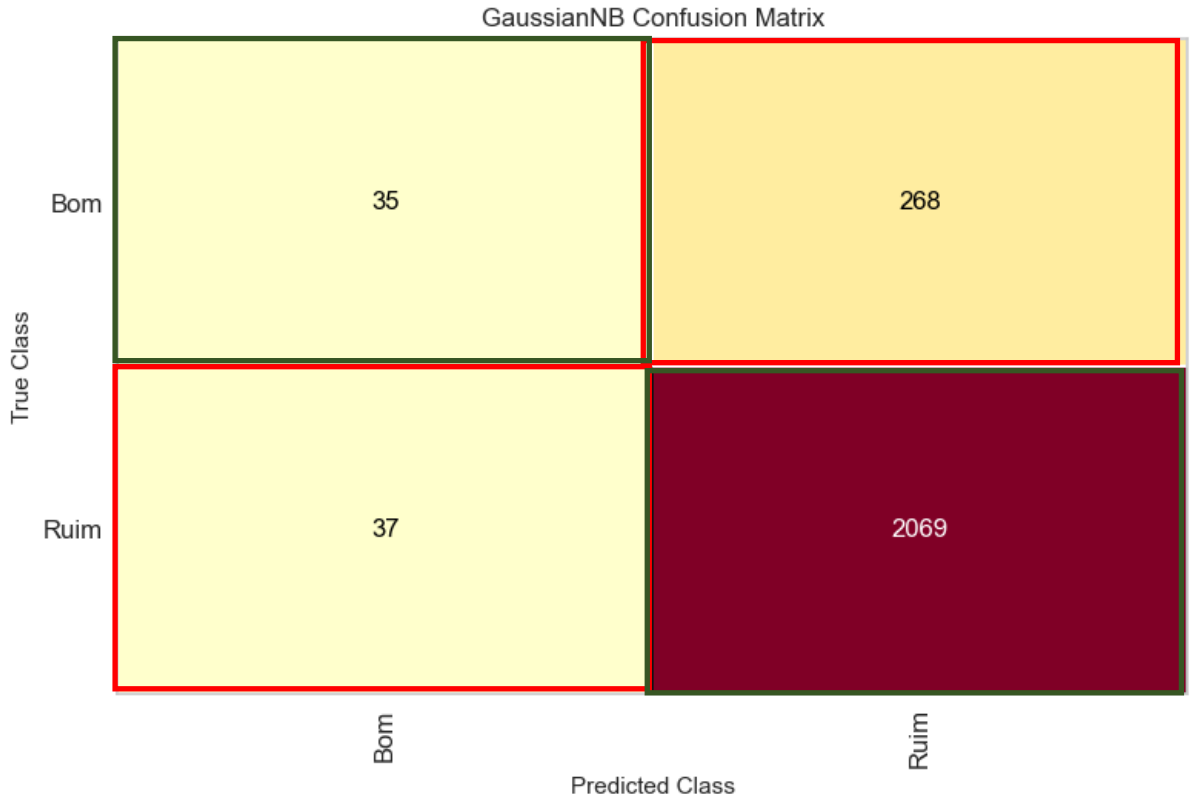
```
# Geração da matriz de confusão e cálculo da taxa de acerto e erro
confusao = confusion_matrix(y_teste, previsoes)
confusao
```

```
array([[  35,  268],
       [  37, 2069]], dtype=int64)
```

Podemos ver no gráfico de visualização da matriz de confusão que temos alguns erros nos quadrantes x=Bom e y= Ruim de 37 erros e no x=Ruim e y=Bom temos 268 erros.

Tivemos 35 acertos no quadrante x= Bom e Y=Bom e 2069 acertos onde x=Ruim e y=Ruim

```
#Visualização da matriz de confusão
v = ConfusionMatrix(GaussianNB())
v.fit(x_treinamento,y_treinamento)
v.score(x_teste, y_teste)
v.poof()
```



Foi realizado o cálculo da taxa de acerto, que foi de 0.87 (87%) que pode ser considerado um bom resultado, que faz com que esse modelo seja válido para uso, nesse caso em questão.

```
#Taxade acerto
taxa_acerto = accuracy_score(y_teste,previsoes)
taxa_erro = 1 - taxa_acerto
taxa_acerto
```

```
0.8733914487339145
```

Devido aos otimos resultados obtidos pelo modelo de Naive Bayes, foi realizado o teste com a carga de teste, que era o dataset da HBO.

Dataset da HBO

```
dfResult
```

	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb_votes	tmdb_score
0	The Wizard of Oz	MOVIE	1939	102	fantasy, family	US	8.1	389774.0	7.6
1	Citizen Kane	MOVIE	1941	119	drama	US	8.3	433804.0	8.0
2	Casablanca	MOVIE	1942	102	drama, romance, war	US	8.5	558849.0	8.2
3	The Big Sleep	MOVIE	1946	116	thriller, crime	US	7.9	84494.0	7.7
4	The Maltese Falcon	MOVIE	1941	100	thriller, romance, crime	US	8.0	156603.0	7.8
...	...	...	...	...	...	...	...	...	...
3257	Covid Diaries NYC	MOVIE	2021	40	documentation	US	3.7	184.0	5.9
3275	Breathless	MOVIE	2021	106	crime, drama, thriller	DO	6.3	27.0	5.9
3279	Furry Friends Forever: Elmo Gets a Puppy	MOVIE	2021	26	animation	US	6.8	14.0	10.0
3283	Marlon Wayans: You Know What It Is	MOVIE	2021	58	comedy	US	3.8	224.0	5.4
3290	Algo Azul	MOVIE	2021	90	comedy	PA	5.9	50.0	2.0

2729 rows × 9 columns

Foi criado uma lista chamada dfNovaCarga, depois os campos categóricos foram transformados em atributos numéricos.

Logo em seguida realizado as previsões de com esses novos registros, usando o modelo de Naive Bayes.

Em seguida atribuído essas previsões ao final do dataset, uma coluna chamada Status.

```
# Previsão com novo registro, transforma os atributos categóricos em numéricos.
dfNovaCarga = dfNovaCarga.iloc[:,0:9].values
dfNovaCarga
```

```
array([['The Wizard of Oz', 'MOVIE', 1939, ..., 8.1, 389774.0, 7.6],
       ['Citizen Kane', 'MOVIE', 1941, ..., 8.3, 433804.0, 8.0],
       ['Casablanca', 'MOVIE', 1942, ..., 8.5, 558849.0, 8.2],
       ...,
       ['Furry Friends Forever: Elmo Gets a Puppy', 'MOVIE', 2021, ...,
        6.8, 14.0, 10.0],
       ['Marlon Wayans: You Know What It Is', 'MOVIE', 2021, ..., 3.8,
        224.0, 5.4],
       ['Algo Azul', 'MOVIE', 2021, ..., 5.9, 50.0, 2.0]], dtype=object)
```

```
# Transformação dos atributos categóricos em atributos numéricos, passando o índice de cada coluna categórica
dfNovaCarga[:,0] = labelencoder1.fit_transform(dfNovaCarga[:,0])

dfNovaCarga[:,1] = labelencoder1.fit_transform(dfNovaCarga[:,1])

dfNovaCarga[:,4] = labelencoder1.fit_transform(dfNovaCarga[:,4])


dfNovaCarga[:,5] = labelencoder1.fit_transform(dfNovaCarga[:,5])
```

```
result=naive_bayes.predict(dfNovaCarga)
result
```

```
array(['Bom', 'Bom', 'Bom', ..., 'Ruim', 'Ruim', 'Ruim'], dtype='<U4')
```

```
dfResult['Status'] = result.tolist()
```

```
dfResult
```

	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb_votes	tmdb_score	Status
0	The Wizard of Oz	MOVIE	1939	102	fantasy, family	US	8.1	389774.0	7.6	Bom
1	Citizen Kane	MOVIE	1941	119	drama	US	8.3	433804.0	8.0	Bom
2	Casablanca	MOVIE	1942	102	drama, romance, war	US	8.5	558849.0	8.2	Bom
3	The Big Sleep	MOVIE	1946	116	thriller, crime	US	7.9	84494.0	7.7	Bom
4	The Maltese Falcon	MOVIE	1941	100	thriller, romance, crime	US	8.0	156603.0	7.8	Bom
...	...	...	...	...	...	...	...	...	...	...
3257	Covid Diaries NYC	MOVIE	2021	40	documentation	US	3.7	184.0	5.9	Ruim
3275	Breathless	MOVIE	2021	106	crime, drama, thriller	DO	6.3	27.0	5.9	Ruim
3279	Furry Friends Forever: Elmo Gets a Puppy	MOVIE	2021	26	animation	US	6.8	14.0	10.0	Ruim
3283	Marlon Wayans: You Know What It Is	MOVIE	2021	58	comedy	US	3.8	224.0	5.4	Ruim
3290	Algo Azul	MOVIE	2021	90	comedy	PA	5.9	50.0	2.0	Ruim

2729 rows × 10 columns

Árvore de decisão

Modelo de aprendizado supervisionado, para problemas de classificação, funciona como um fluxograma de maneira recursiva, analisando cada novo nó, composto por nós, nós-raiz e nós-folhas.

Foi utilizado o índice GINI, que verifica a distribuição dos dados nas variáveis preditoras de acordo com a variação da variável target, porém com um método diferente. A variável preditora com o menor índice Gini será a escolhida para o nó principal da árvore, pois um baixo valor do índice indica maior ordem na distribuição dos dados.

Vantagens

- Fácil de entender: A visualização de uma árvore de decisão torna o problema fácil de compreender, mesmo para pessoas que não tenham perfil analítico. Não requer nenhum conhecimento estatístico para ler e interpretar. Sua representação gráfica é muito intuitiva e permite relacionar as hipóteses também facilmente.

- Útil em exploração de dados: A árvore de decisão é uma das formas mais rápidas de identificar as variáveis mais significativas e a relação entre duas ou mais variáveis. Com a ajuda de árvores de decisão, podemos criar variáveis/características que tenham melhores condições de predizer a variável alvo.

- Menor necessidade de limpar dados: Requer menos limpeza de dados em comparação com outras técnicas de modelagem. Até um certo nível, não é influenciado por pontos fora da curva “outliers” nem por valores faltantes (“missing values”).

- Não é restrito por tipos de dados: Pode manipular variáveis numéricas e categóricas.

- Método não paramétrico: A árvore de decisão é considerada um método não-paramétrico. Isto significa que as árvores de decisão não pressupõem a distribuição do espaço nem a estrutura do classificador.

Desvantagens

- Sobreajuste (“Over fitting”): Sobreajuste é uma das maiores dificuldades para os modelos de árvores de decisão. Este problema é resolvido através da definição de restrições sobre os parâmetros do modelo e da poda (discutido em mais detalhes abaixo).

- Não adequado para variáveis contínuas: ao trabalhar com variáveis numéricas contínuas, a árvore de decisão perde informações quando categoriza variáveis em diferentes categorias.

Foi preciso separar as previsões da classificação (Status)

```
# Separar as previssões da classificação (Status)
previsores = dataSet.iloc[:,0:9].values
classe = dataSet.iloc[:,9].values
```

Tivemos que transformar os atributos categóricos em atributos numéricos, conforme o código abaixo e criar as variáveis de treinamento.

```
# Transformação dos atributos categóricos em atributos numéricos, passando o índice de cada coluna categórica
labelencoder1 = LabelEncoder()
previsores[:,0] = labelencoder1.fit_transform(previsores[:,0])

labelencoder2 = LabelEncoder()
previsores[:,1] = labelencoder1.fit_transform(previsores[:,1])

labelencoder3 = LabelEncoder()
previsores[:,4] = labelencoder1.fit_transform(previsores[:,4])

labelencoder4 = LabelEncoder()
previsores[:,5] = labelencoder1.fit_transform(previsores[:,5])
```

```
x_treinamento, x_teste,y_treinamento, y_teste = train_test_split(previsores,classe,test_size= 0.3, random_state=0)
x_teste
```

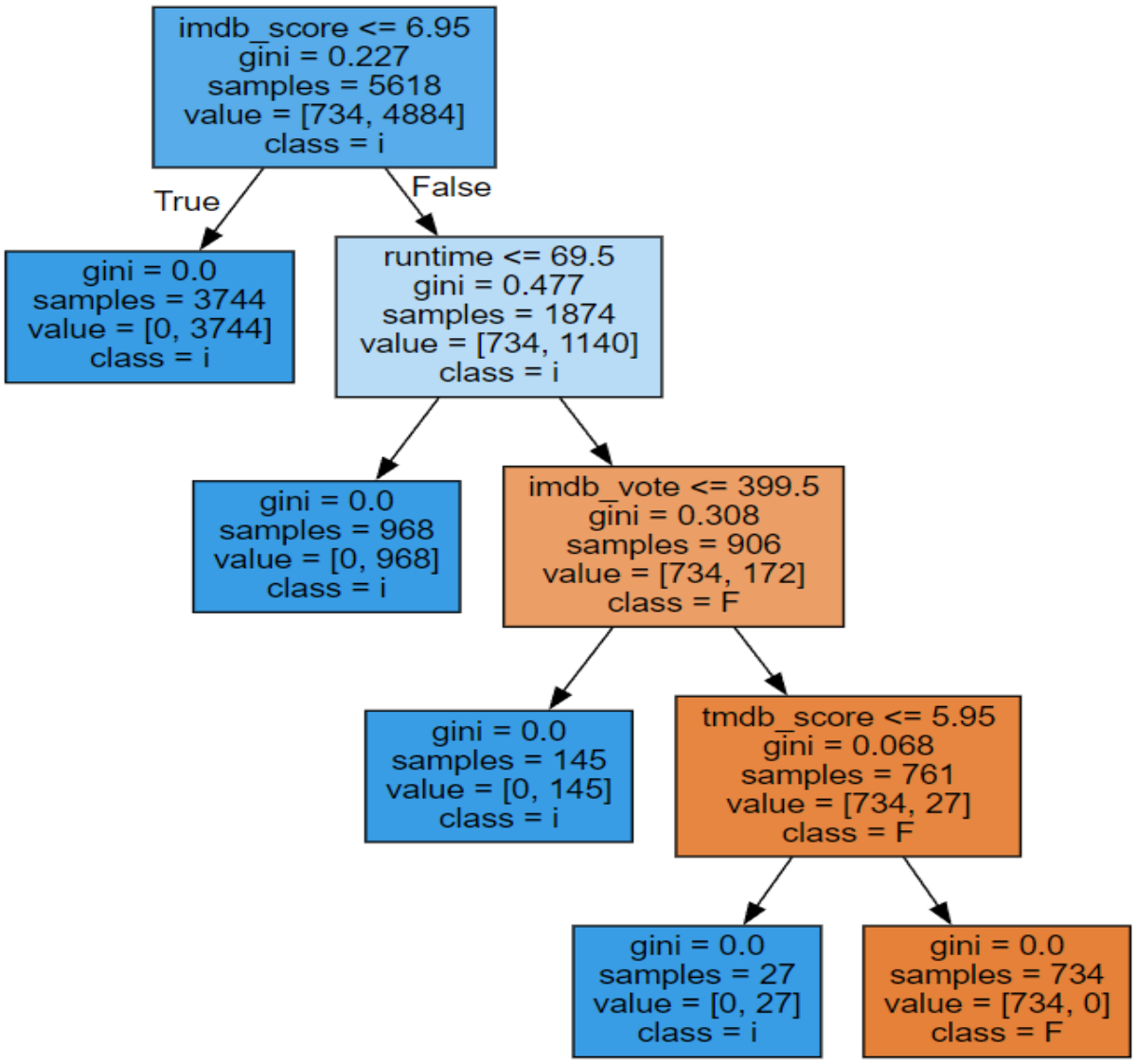
Depois foi criado e realizado o treinamento do modelo de Árvore de Decisão e exportamos a arvore para analisar psoteriormente no site https://dreampuf.github.io/GraphvizOnline/

```
#Criação e treinamento do modelo - Árvore de decisão
arvore = DecisionTreeClassifier(
                                max_depth = None,
                                max_features = None,
                                min_samples_leaf=1,
                                min_samples_split=2
)
arvore.fit(x_treinamento, y_treinamento)
```

```
DecisionTreeClassifier()
```

```
# Exportação da árvore de decisão para o formato .dot, para posterior visualização
export_graphviz(arvore, out_file='tree.dot')
arquivo = 'tree.dot'
```

A árvore gerada pelo modelo.



Abaixo temos os atributos da Árvore de Decisão, quanto maior o valor, maior o ganho de informação do atributo para o modelo, os melhores atributos são runtime e imdb_score.

```
#atributos da árvore de decisão, os que tem maior valor representam maior ganho de informação. Importante.
arvore.feature_importances_
```

```
array([0.48137077, 0.        , 0.30025252, 0.17756497, 0.04081175])
```

```
#Atributos da árvore
#runtime    genres    imdb_score    imdb_votes    tmdb_score
print('Runtime    X[0]-->',arvore.feature_importances_[0])
print('genres     X[1]-->',arvore.feature_importances_[1])
print('imdb_score X[2]-->',arvore.feature_importances_[2])
print('imdb_votes X[3]-->',arvore.feature_importances_[3])
print('tmdb_score X[4]-->',arvore.feature_importances_[4])
```

```
Runtime    X[0]--> 0.48137076696590986
genres     X[1]--> 0.0
imdb_score X[2]--> 0.30025251796863334
imdb_votes X[3]--> 0.1775649679462195
tmdb_score X[4]--> 0.04081174711923726
```

Foi usado comando predict para se obter as previsões

```
# obtenção das previsões
previsoes=arvore.predict(x_teste)
previsoes
```

```
array(['Ruim', 'Ruim', 'Ruim', ..., 'Ruim', 'Ruim', 'Ruim'], dtype=object)
```

Cálculamos a matriz de confusão, e podemos ver que não apresenta erros em nenhum dos quadrantes, assim como mostra o cálculo da taxa de acerto de 100% e a taxa de erro de 0%.

```
# matriz de confusão - mostra os acertos e os erros.
confusao = confusion_matrix(y_teste, previsoes)
confusao
```

```
array([[ 303,    0],
       [   0, 2106]], dtype=int64)
```

```
# taxa de acerto - é superior ao modelo de Naive Bayers
taxa_acerto = accuracy_score(y_teste, previsoes)
taxa_acerto
```

```
1.0
```

```
# taxa de erro
taxa_erro = 1 - taxa_acerto
taxa_erro
```

```
0.0
```

Random Forest

Consiste em gerar vários modelos, evitando assim o overfitting.

Cria de forma aleatória várias Árvores de Decisão e combina o resultado de todas elas para chegar no resultado.

Muito usado no setor bancário, mercado financeiro, Hospitalar e comércio eletrônico.

Vantagens

- Retorna de maneira muito compreensiva a importância atribuída para cada variável independente.
- Pode ser usada para regressão e para classificação.

- Muito fácil de implementar, geralmente produz bons resultados
- Se houver árvores suficiente na floresta, o classificador não irá sobre ajustar o modelo e gerar overfitting.

Desvantagens

- Quantidade grande de árvores pode deixar o algoritmo lento e ineficiente para predições em tempo real.
- Para maior acurácia, precisa de mais árvores, o que faz o modelo ficar mais lento.
- Ferramenta de modelagem preditiva e não descritiva.

Foi realizado a separação das variáveis previsoras e da classificadoras, os campos categóricos foram transformados em númericos e foram criados as variáveis de treinamento igual da Árvore de Decisão.

```
# Separar as previssões da classe (Status)
previsores = dataSet.iloc[:,0:5].values
classe = dataSet.iloc[:,5].values

# Transformação dos atributos categóricos em atributos numéricos, passando o índice de cada atributo categórico
labelencoder = LabelEncoder()
previsores[:,1] = labelencoder.fit_transform(previsores[:,1])


# Divisão da base de dados entre treinamento e de teste (30% para testar e 70% para treinar)
x_treinamento, x_teste,y_treinamento, y_teste = train_test_split(previsores,classe,test_size= 0.3, random_state=1)
x_teste
```

Foi feito o treinamento do modelo, conforme imagem abaixo.

```
# Criação do modelo, treinamento, obtenção das previsões e da taxa de acerto
floresta = RandomForestClassifier(n_estimators=100)
floresta.fit(x_treinamento, y_treinamento)
```

Foi gerado os atributos e seus respectivos valores de significancia(importância de cada um dos atributos, tem o maior ganho de informação), temos na imagem abaixo que o runtime e o imdb_Score tem um peso maior.

```
#atributos da rondom forest
floresta.feature_importances_
```

```
array([0.41916018, 0.01016049, 0.3418108 , 0.12870302, 0.10016552])
```

```
#Atributos da árvore
#runtime    genres    imdb_score    imdb_votes    tmdb_score
print('Runtime    X[0]-->',floresta.feature_importances_[0])
print('genres     X[1]-->',floresta.feature_importances_[1])
print('imdb_score X[2]-->',floresta.feature_importances_[2])
print('imdb_votes X[3]-->',floresta.feature_importances_[3])
print('tmdb_score X[4]-->',floresta.feature_importances_[4])
```

```
Runtime    X[0]--> 0.41916017997266297
genres     X[1]--> 0.010160491695535602
imdb_score X[2]--> 0.34181079540606324
imdb_votes X[3]--> 0.12870301563783865
tmdb_score X[4]--> 0.10016551728789942
```

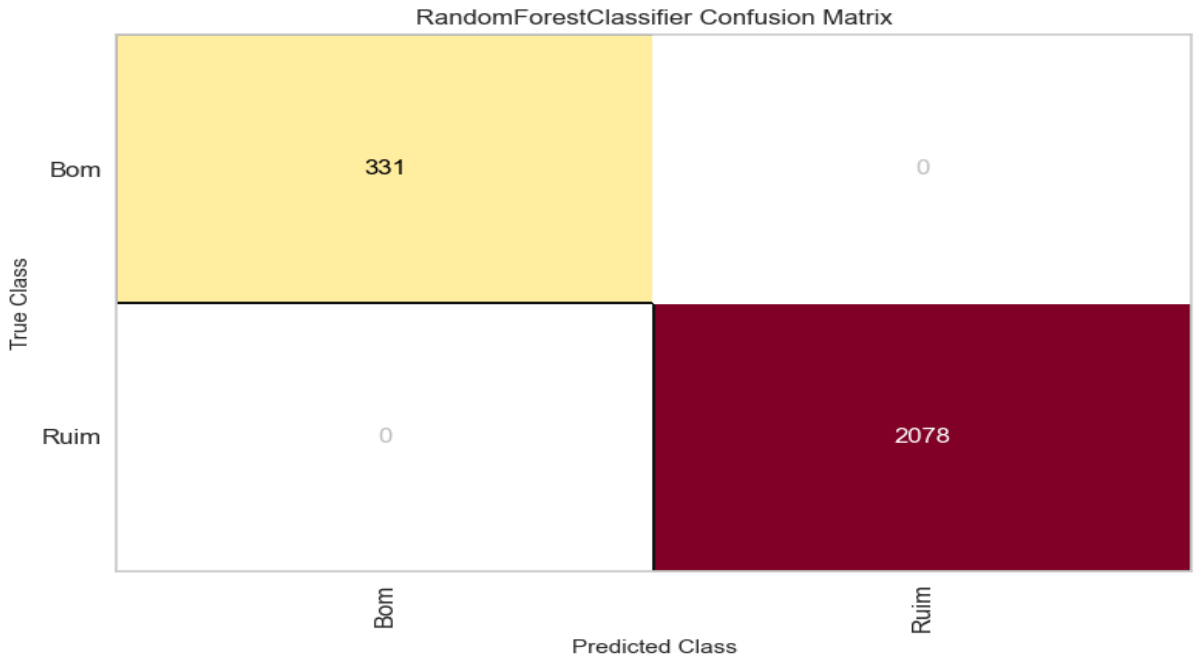
Cálculamos a acurácia com o comando abaixo

```
# Acurácia
print('Acurácia: %.4f' %accuracy_score(y_teste,previsoes))
```

```
Acurácia: 1.0000
```

Pela matriz de confusão mostrada abaixo podemos ver que ele apresenta uma taxa de sucesso de 100% e conseguiu melhores resultados que a Árvore de Decisão no quadrante “Bom”

```
#Visualização da matriz de confusão
v = ConfusionMatrix(RandomForestClassifier())
v.fit(x_treinamento,y_treinamento)
v.score(x_teste, y_teste)
v.poof()
```

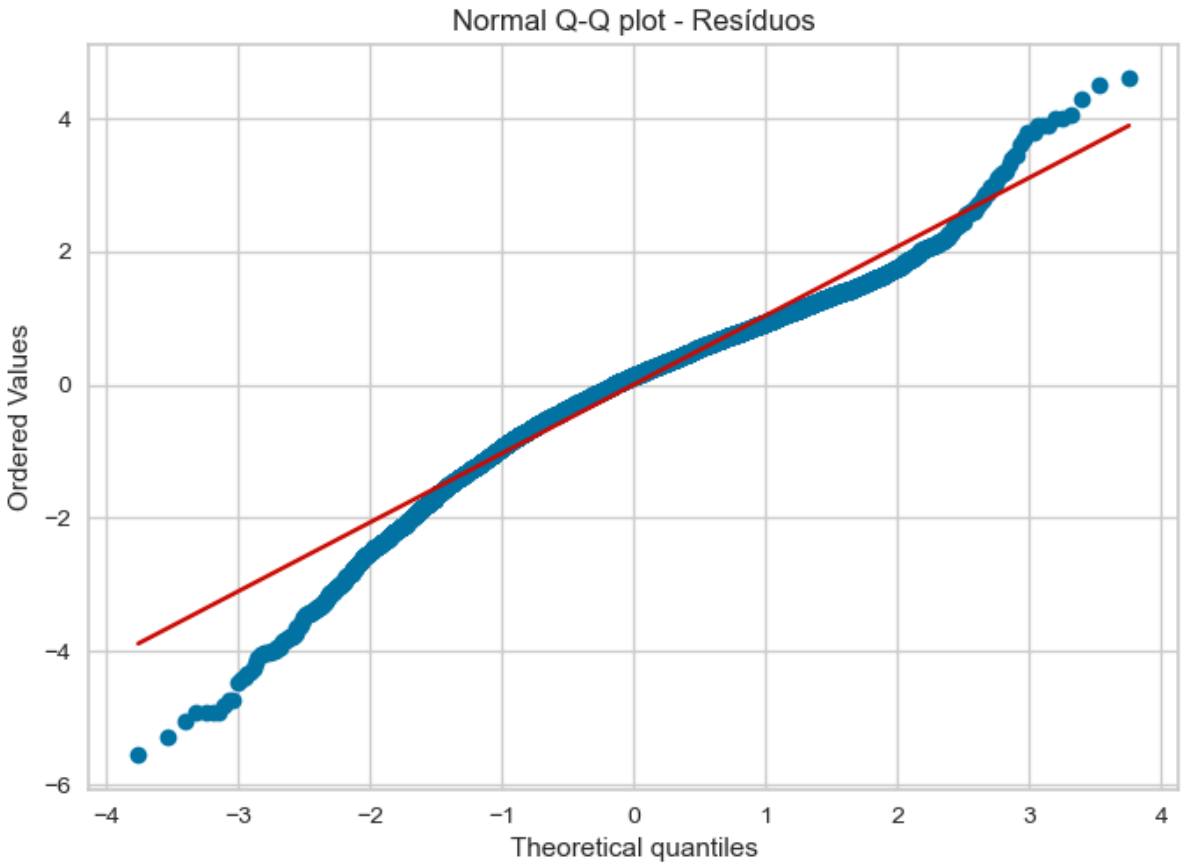


Abaixo podemos ver a representação da árvore gerada, pelo modelo.

6. Interpretação dos Resultados

Após rodar os 3 modelos de machine learning Regressão Linear, Naive Bayes e Arvore de decisão, podemos verificar que tanto a Árvore de Decisão como o modelo de Naive Bayes seriam os mais indicados para serem implantados, sendo o modelo de regressão Linear o não recomendado, como veremos abaixo.

No modelo de Regressão Linear conforme a figura abaixo, do gráfico da Normal Q-Q plot, foi interpretado que a regressão linear seria indicada nesse caso, a partir dos dados que temos, uma vez o gráfico da Normal Q-Q, parece demonstrar ser uma distribuição normal, com apenas uma grande divergência nas extremidades, se os dados forem dimensionaliza-dos, poderia ser uma das partes com maior aderência ao modelo de regressão linear.



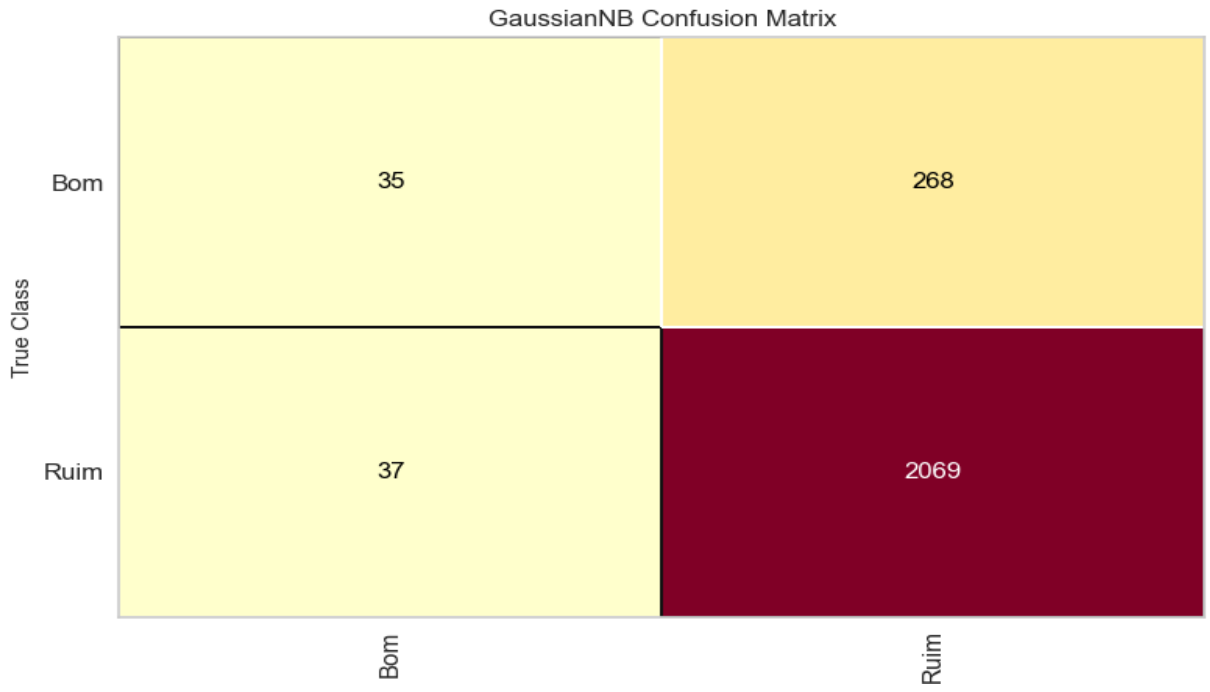
Porém pelos resultados dos cálculos estatisticos usado, tanto Shapiro-Wilk com uma distribuição de dados menos que 5 mil registros, assim como o teste de Lilliefors e o de Anderson, infelizmente ambos não recomendam o uso de regressão linear pois o p-valor está muito longe de 0,05 ou 5%.

Além disso o valor obtido pela raiz quadratica é muito baixo 32%, representando que relação entre as 2 grandezas analizadas é muito baixa.

```
# Regressão usando as grandezas runtime e imdb_score
regressao = smf.ols("imdb_score ~ tmdb_score", data =dfRegressao).fit()
print(regressao.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.324
Model:                            OLS   Adj. R-squared:                  0.324
Method:                 Least Squares   F-statistic:                     3841.
Date:                Wed, 01 Feb 2023   Prob (F-statistic):               0.00
Time:                        13:34:08   Log-Likelihood:                -11807.
No. Observations:                8027   AIC:                         2.362e+04
Df Residuals:                    8025   BIC:                         2.363e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.7694      0.058     47.817      0.000       2.656       2.883
tmdb_score     0.5364      0.009     61.974      0.000       0.519       0.553
==============================================================================
Omnibus:                      880.065   Durbin-Watson:                   1.903
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1772.540
Skew:                          -0.703   Prob(JB):                         0.00
Kurtosis:                       4.823   Cond. No.                         33.7
==============================================================================
```

Por outro lado, o gráfico da matriz de confusão do modelo de Naive Bayes demonstrou uma alta quantidade de acertos como poucos erros, conforme a imagem abaixo.



Além disso os cálculos da taxa de acerto apresentam um resultado de 87% porcento, conforme a imagem abaixo.

```
#Taxade acerto
taxa_acerto = accuracy_score(y_teste,previsoes)
taxa_erro = 1 - taxa_acerto
taxa_acerto
```

```
0.8733914487339145
```

Obtivemos também um ótimo resultado da Árvore de Decisão, que apresenta 0 erros e com uma taxa de acerto de 100% na classificação.

```
# matriz de confusão - mostra os acertos e os erros.
confusao = confusion_matrix(y_teste, previsoes)
confusao
```

```
array([[ 303,    0],
       [   0, 2106]], dtype=int64)
```

```
# taxa de acerto - é superior ao modelo de Naive Bayers
taxa_acerto = accuracy_score(y_teste, previsoes)
taxa_acerto
```

```
1.0
```

7. Apresentação dos Resultados

Title: ANÁLISE DE PROGRAMAS DE SERVIÇOS DE STREAMING

1 Problem Statement
What problem are you trying to solve?
What larger issues do the problem address?

Analisar a relação das 2 grandezas de pontuação
(IMDB e TMDB) e classificar os programas dos serviços de
streaming

Foi usado o modelo de machine learn para prever a pontuação tmdb_score ou classificar os programas entre "Bom"e "Ruim"

2 Outcomes/Predictions
What prediction(s) are you trying to make?
Identify applicable predictor (X) and/or target (y) variables.

As variáveis preditoras: imdb_score, runtime, imdb_votes, tmdb_score

As variáveis de previsão: tmdb_score e classificação dos programas entre "Bom"e "Ruim".

3 Data Acquisition
Where are you sourcing your data from?
Is there enough data? Can you work with it?

Os dados foram extraídos do kaggle sendo 3 bases 2 para análise e treinamento do modelo de machine learning e uma para ser usada como carga de teste

4 Modeling
What models are appropriate to use given your outcomes?

Não foi possível determinar uma relação entre as 2 grandezas analisadas, a partir da pontuação da IMDB não tem muita relação com a pontuação TMDB.

Foi verificado a classificação dos programas das streaming para pelo menos determinar quais programas poderiam ser classificados como "Bom" ou "Ruim"

5 Model Evaluation
How can you evaluate your model's performance?

Dependendo do modelo, podemos avaliar os coeficientes, correlação das grandezas , matriz de confusão, testes de estatística e taxa de acerto, dos modelos analisados.

6 Data Preparation
What do you need to do to your data in order to run your model and achieve your outcomes?

Foi excluído as linhas com valores null, alguns dados precisou ser corrigido, foi criado a coluna de classificação, os dadasets foram concatenados em apenas 1 dataset, mantendo um separado para ser usado como carga de teste.

Analisando os modelos temos os resultados abaixo:

Modelos	R-Quadrática	P-Valor	Precisão	Correlação	Bom	Ruim
Regressão Linear	32%	0,09%		57%		
Naive Bayes			88%		30	2088
Árvire de Descião			100%		303	2106
Randon Forest			100%		331	2078

8. Links

Link para o vídeo: https://youtu.be/iqKyEjH5BT8

Link para o repositório: https://github.com/brennoneves/PUC.git

https://drive.google.com/drive/folders/1lbSnChDuxyozvF5t8bUE_Agbo8nysihW?usp=share_link

APÊNDICE

Programação/Scripts

```
#Importação das bibliotecas
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import statsmodels
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from statsmodels.stats.diagnostic import lilliefors
from apyori import apriori

# Modelo de Naive Bayes
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
from yellowbrick.classifier import ConfusionMatrix
#Modelo de Arvore de decisão
from sklearn.tree import DecisionTreeClassifier
import graphviz
from sklearn.tree import export_graphviz


# Random Forest
```

```
from sklearn.ensemble import RandomForestClassifier


#versão do python
!python --version


# Carregar os dados das operadoras de Streaming
dNetflix = pd.read_csv('PUC/Netflix.csv')
dAmazon = pd.read_csv('PUC/amazon_prime.csv')
dHbo = pd.read_csv('PUC/HBO.csv')


# Selecionar quais campos iremos usar das 3 bases carregadas
dfNetflix = dNet-
flix[['title','type','release_year','runtime','genres','production_countries','imdb_score',

'imdb_votes','tmdb_score']]

dfNetflix= dfNetflix.dropna()

dfNetflix['streaming'] = 'Netflix'

dfAma-
zon=dAmazon[['title','type','release_year','runtime','genres','production_countries','imdb_sc
ore',

 'imdb_votes','tmdb_score']]

dfAmazon= dfAmazon.dropna()

dfAmazon['streaming'] = 'Amazon'

dfH-
bo=dHbo[['title','type','release_year','runtime','genres','production_countries','imdb_score',

'imdb_votes','tmdb_score']]

dfHbo = dfHbo.dropna()

dfHbo['streaming'] = 'HBO'

# Visualizar o dataset dfHbo
dfHbo


# Verificar tamanho do dataset
dfHbo.shape
```

```
#Visualizar o dataset dfAmazon
dfAmazon


#Verificar o tamanho do dataset
dfAmazon.shape


#Verificar o dataset dfNetflix
dfNetflix


#Visualizar o tamanho do dataset
dfNetflix.shape


# Nova carga de teste
dfNovaCarga =
dHbo[["title","type","release_year","runtime","genres","production_countries","imdb_score
","imdb_votes",
                "tmdb_score"]]

# Remover caractere [ ]
# Remover linhas com índices duplicados
dfNovaCarga = dfNovaCarga[~dfNovaCarga.index.duplicated()]

# Iremos remover as linhas cujo país seja vazio
dfNovaCarga = dfNovaCarga[dfNovaCarga.production_countries !='']

# Remover aspas simples das colunas genres e production_countries
dfNovaCarga["genres"] = dfNovaCarga["genres"].str.replace("'", "")
dfNovaCarga["production_countries"] = dfNovaCar-
ga["production_countries"].str.replace("'", "")

dfNovaCarga["genres"] = dfNovaCarga["genres"].str.replace("]","", regex=True)
dfNovaCarga["genres"] = dfNovaCarga["genres"].str.replace("[","", regex=True)
dfNovaCarga["production_countries"] = dfNovaCar-
ga["production_countries"].str.replace("]","", regex=True)
dfNovaCarga["production_countries"] = dfNovaCar-
ga["production_countries"].str.replace("[","", regex=True)
dfNovaCarga = dfNovaCarga.dropna()

# Iremos remover as linhas cujo país seja vazio
dfNovaCarga = dfNovaCarga[dfNovaCarga.production_countries !='']
dfResult=dfNovaCarga
dfResult
# Juntar as 2 bases e um único Dataset
df1= [dfNetflix,dfAmazon]
```

```
dataSet=pd.concat(df1)


# Remover linhas com indices duplicados
dataSet = dataSet[~dataSet.index.duplicated()]


# Iremos remover as linhas cujo país seja vazio
dataSet = dataSet[dataSet.production_countries !='']


#Visualizar o tamanho do dataset
dataSet.shape


#Verificar as informações sobre os dados, tipos de dados
dataSet.info()


# Remover aspas simples das colunas genres e production_countries
dataSet["genres"] = dataSet["genres"].str.replace("'", "")
dataSet["production_countries"] = dataSet["production_countries"].str.replace("'", "")


# Remover caracter -> [ ]
dataSet["genres"] = dataSet["genres"].str.replace("]","", regex=True)
dataSet["genres"] = dataSet["genres"].str.replace("[","", regex=True)
dataSet["production_countries"] = dataSet["production_countries"].str.replace("]","", re-
gex=True)
dataSet["production_countries"] = dataSet["production_countries"].str.replace("[","", re-
gex=True)


#Deletar as linhas que contém algum elemento vazio
dataSet = dataSet.dropna()


# Ordenar os valores da coluna genres
dataSet["genres"] = [','.join(sorted(x.split(','))) for x in dataSet['genres']]


# Remover a parte de European dos generos
dataSet["genres"] = dataSet["genres"].str.replace("european",'')




# Converter o nome do País para a sigla
dataSet['production_countries'] = dataSet['production_countries'].replace("['Lebanon']",'LB')
```

```
dataSet['production_countries'] = dataSet['production_countries'].replace("['United States of America']",'US')


# Ordenar o dataset pelo ano de lançamento
dataSet = dataSet.sort_values(by='release_year')


# Eliminar as linhas que não tem pais
dataSet = dataSet[dataSet.production_countries !='']


# Resetar os indexs
dataSet.reset_index()


# Verificar se tem algum null nos dados
dataSet.isnull().sum()


#verificar as descrições do dataset
dataSet.describe()


#Verificar tamanho do dataset
dataSet.shape


# Separa os gêneros por coluna.
dataSet["action"] = np.where(dataSet["genres"].str.contains('action'), 'action' , np.nan )
dataSet["comedy"] = np.where(dataSet["genres"].str.contains('comedy'), 'comedy' , np.nan )
dataSet["thriller"] = np.where(dataSet["genres"].str.contains('thriller'), 'thriller' , np.nan )
dataSet["drama"] = np.where(dataSet["genres"].str.contains('drama'), 'drama' , np.nan )
dataSet["family"] = np.where(dataSet["genres"].str.contains('family'), 'family' , np.nan )
dataSet["romance"] = np.where(dataSet["genres"].str.contains('romance'), 'romance' , np.nan )
dataSet["animation"] = np.where(dataSet["genres"].str.contains('animation'), 'anima-
tion' , np.nan )
dataSet["fantasy"] = np.where(dataSet["genres"].str.contains('fantasy'), 'fantasy' , np.nan )
dataSet["reality"] = np.where(dataSet["genres"].str.contains('reality'), 'reality' , np.nan )
```

```
dataSet["crime"] =        np.where(dataSet["genres"].str.contains('crime'),        'crime'        ,
np.nan  )
dataSet["music"] =        np.where(dataSet["genres"].str.contains('music'),        'music'        ,
np.nan  )
dataSet["sport"] =        np.where(dataSet["genres"].str.contains('sport'),        'sport'        ,
np.nan  )
dataSet["history"] =      np.where(dataSet["genres"].str.contains('history'),      'history'      ,
np.nan  )
dataSet["documentation"] = np.where(dataSet["genres"].str.contains('documentation'),
'documentation' ,    np.nan  )
dataSet["horror"] =       np.where(dataSet["genres"].str.contains('horror'),       'horror'       ,
np.nan  )
dataSet["scifi"] =        np.where(dataSet["genres"].str.contains('scifi'),        'scifi'        ,
np.nan  )
dataSet["war"] =          np.where(dataSet["genres"].str.contains('war'),          'war'          ,
np.nan  )
dataSet["western"] =      np.where(dataSet["genres"].str.contains('western'),      'western'
,    np.nan  )


#Verificar se existe algum valor null
dataSet.isnull().sum()


# Classificação nos gostos do usuário BRENNO
#imdb >= 7 a pontuação tmdb >=6 o número de runtime maior que 70 e quantidade de
votos deve ser maior que 400
filtro = [
   (dataSet['imdb_score'] >= 7) & (dataSet['tmdb_score'] >= 6) & (dataSet['runtime'] >= 70) &
(dataSet['imdb_votes'] >= 400)
]

#Resultados
resultado = ['Bom']

#Novo coluna com o status de bom e 0
dataSet['Status'] = np.select(filtro, resultado)

#Tudo que for 0 substitui para ruim
dataSet['Status'] = dataSet['Status'].str.replace("0",'Ruim')

#Verificar se todos os dados estão divididos entre "BOM" e "RUIM"
dataSet['Status'].unique()



# Quantidade de shows e movie produzidos no ano, por tipo
```

```
dfTiposShow=dataSet.groupby(['release_year','type']).size().reset_index(name='Total')
dfTiposShow


# Quantidade de programas produzidos no ano por streaming
dfStreaming=dataSet.groupby(['release_year','streaming']).size().reset_index(name='Total')
dfStreaming


# Gráfico de plot analisando por ano de lançamento, acessos e pontuação dos programas
dfBox=dataSet[["streaming","runtime","imdb_score","tmdb_score"]]
dfBox.plot(kind="box",figsize=(10,6),subplots=True)


# Gráfico de plot melhorado, sobre a pontuação do imdb
grafico = px.box(dataSet, y = "imdb_score")
grafico.show()


# Gráfico de plot melhorado, sobre a pontuação do tmdb
grafico = px.box(dataSet, y = "tmdb_score")
grafico.show()


# Gráfico de plot melhorado, sobre os acessos
grafico = px.box(dataSet, y = "runtime")
grafico.show()


#Verificando os possíveis outliers - acessos
runtime_maior =dataSet[dataSet['runtime'] > 184]
runtime_maior = runtime_maior.reset_index()
runtime_maior[["title","streaming","release_year","imdb_score","tmdb_score"]]


#Verificando os possíveis outliers - tmdb
piores_notas_tmdb=dataSet[dataSet['tmdb_score'] < 3.4]
piores_notas_tmdb=piores_notas_tmdb.reset_index()
piores_notas_tmdb[["title","streaming","release_year","imdb_score","tmdb_score"]]


#Verificando os possíveis outliers - tmdb
maiores_notas_tmdb=dataSet[dataSet['tmdb_score'] > 9.8]
maiores_notas_tmdb=maiores_notas_tmdb.reset_index()
maiores_notas_tmdb[["title","streaming","release_year","imdb_score","tmdb_score"]]
```

```
#Verificando os possíveis outliers - imdb
piores_notas_imdb=dataSet[dataSet['imdb_score'] < 3]
piores_notas_imdb=piores_notas_imdb.reset_index()
piores_notas_imdb[["title","streaming","release_year","imdb_score","tmdb_score"]]


#Verificando os possíveis outliers - imdb
maiores_notas_imdb=dataSet[dataSet['imdb_score'] > 9.5]
maiores_notas_imdb=maiores_notas_imdb.reset_index()
maiores_notas_imdb[["title","streaming","release_year","imdb_score","tmdb_score"]]


# Analisando a tabulação cruzada entre os campos do Dataset (type - streaming)
pd.crosstab(dataSet["type"], dataSet["streaming"])


# Gráfico de barra, agrupando por streaming
dataSet['streaming'].value_counts().plot(kind='bar')


#parâmetros do gráfico de pizza
kwargs = dict(
   startangle = 90,
   colormap = 'Pastel2',
   fontsize = 13,
   explode = None,
   figsize=(60,5),
   autopct = '%1.1f%%',
   title = 'Streaming'
 )


#Gerar o gráfico de pizza
dataSet['streaming'].value_counts().plot.pie(**kwargs)


#parâmetros do gráfico de pizza
kwargs = dict(
   startangle = 90,
   colormap = 'Pastel2',
   fontsize = 13,
   explode = None,
   figsize=(60,5),
   autopct = '%1.1f%%',
   title = 'Type'
 )
```

```
#Gerar o gráfico de pizza
dataSet['type'].value_counts().plot.pie(**kwargs)


# Gráfico de histograma da pontuação - IMDB
sns.histplot(dataSet,x= "imdb_score", bins=10, color="orange", kde=True,stat="probability")


# Gráfico de histograma da pontuação - TMDB
sns.histplot(dataSet,x= "tmdb_score", bins=10, color="orange", kde=True,stat="probability")


# Gráfico de dispersão com base nos países, pontuação do imdb e o tipo.
fig = px.scatter(dataSet, x = "production_countries",y = "imdb_score" , symbol = 'streaming',
color = "type",
            hover_name = "type", log_x = False, width = 800)
fig.update_traces(marker=dict(size = 12,line = dict(width = 2)),selector=dict(mode = 'mark-
ers'))
fig.update_layout(title = 'Tipo de produção por país')
fig.update_xaxes(title = 'Paises')
fig.update_yaxes(title = 'Pontuação')
fig.show()


# Correlação entre as pontuações de IMDB e TMDB
plt.scatter(dataSet.imdb_score,dataSet.tmdb_score)
plt.title("Correlação")
plt.xlabel("IMDB")
plt.ylabel("TMDB")
plt.grid(True)
plt.show()


# Criar um Dataframe com apenas os campos nessa análise
dfRegressao = dataSet[["streaming","runtime","imdb_score","imdb_votes","tmdb_score"]]
dfRegressao


# Avaliar as correlações das grandezas
dataSet.corr()


# Correlação das pontuações (Amazon)
dfNet = dataSet.where(dataSet["streaming"]=="Amazon")
dfNet = dfNet.dropna()
dfNet[["imdb_score","tmdb_score"]].corr()
```

```
# Correlação das pontuações (Netflix)
dfNet = dataSet.where(dataSet["streaming"]=="Netflix")
dfNet = dfNet.dropna()
dfNet[["imdb_score","tmdb_score"]].corr()


# Regressão usando as grandezas runtime e imdb_score
regressao = smf.ols("imdb_score ~ tmdb_score", data =dfRegressao).fit()
print(regressao.summary())


# Os Coeficientes (Linear/ Angular)
coefs = pd.DataFrame(regressao.params)
coefs.columns = ["Os_Coeficientes"]
print(coefs)


# Previssão para cada um dos valores do imdb com base no valor do tmdb
regressao.predict()


# Residuo é distância entre os dados e a reta ajustada. Mostra a distância que o valor está
da reta ajustada (reta de regressão)
residuos = regressao.resid
residuos



# Gerar o gráfico da Reta de Regressão
plt.scatter(y=dfRegressao.imdb_score,x=dfRegressao.tmdb_score, color='blue', s=10, al-
pha=.3)
X_plot = np.linspace(min(dfRegressao.tmdb_score), max(dfRegressao.tmdb_score),
len(dfRegressao.tmdb_score))
X_plot = np.linspace(1,10)
plt.plot(X_plot, X_plot*regressao.params[1] + regressao.params[0], color='r')
plt.title('Reta de regressão')
plt.ylabel('IMDB')
plt.xlabel('TMDB')
plt.show()


# É uma Distribuíção normal - para ser distribuição normal eles precisam concindir com a
linha vermelha.
# Consegue realizar uma previsão boa nos valores do meio da reta, mas erra muito nas
extremidades
stats.probplot(residuos, dist="norm", plot=plt)
plt.title("Normal Q-Q plot - Resíduos")
plt.show()
```

```
# Teste de Shapiro-Wilk
# Nível de significância de 0,05 ou 5%
# Quando p > 0,05(Distribuição normal)
# Teste de Normalidade dos resíduos - pvalue > 0.05 para ter uma distribuição normal (Para elementos menores que 5000)
stats.shapiro(residuos)


# Teste Lilliefors
#o teste de Shapiro-Wilk teve como resultado W = 0,98105, p = 0,0306.
#Assim, nossos erros não são aleatórios e não há homogeneidade de variância dos resíduos.
statsmodels.stats.diagnostic.lilliefors(dataSet.imdb_score, dist="norm")


# Teste de normalidade de Anderson
ad_stat, ad_critico, ad_teorico = stats.anderson(residuos, 'norm')
print("O Valor da estatística calculada: " ,ad_stat)
print("Os valores : ",ad_critico)
print("Os níveis de significancia: ",ad_teorico)


#Verificar se é uma distribuição normal pelo teste de Anderson
if ad_stat < ad_critico[2]:
   print("Com " + str(100 - ad_teorico[2]) + "% de confiança, os dados são similares a uma distribuição normal"
         +"segundo o teste de AD")
else:
   print("Com " + str(100 - ad_teorico[2]) + "% de confiança, os dados não são similares a uma distribuição normal"
         +"segundo o teste de AD")


# Análise da Homocedasticidade dos resíduos
# Variação constante dos resíduos, além da distruição normal
# Tem que gerar algo parecido com um retangulo
#Ao analisar o gráfico acima, é possível perceber que os resíduos não são homogêneos para todos os valores de Y
# São mais homogeneos nos intervalos de x entre 5 e 7,5. Mais ou menos
plt.scatter(y=residuos, x=regressao.predict(), color='red')
plt.hlines(y=0, xmin=4.5, xmax=8, color='orange')
plt.ylabel('Resíduos')
plt.xlabel('Valores Preditos')
plt.show()
```

```
#Filtrar os campos do dataser que iremos usar no treinamento
dataSet = dataSet[["runtime","genres","imdb_score","imdb_votes","tmdb_score","Status"]]
dataSet


# Separar as previssões da classificação (Status)
previsores = dataSet.iloc[:,0:5].values
classe = dataSet.iloc[:,5].values


#Visualizar os dados da variável previsores
previsores


#Visualizar as classificações
Classe

# Transformação dos atributos categóricos em atributos numéricos, passando o índice de cada coluna categórica
labelencoder1 = LabelEncoder()
previsores[:,1] = labelencoder1.fit_transform(previsores[:,1])


#Criar as variáveis para treinamento
x_treinamento, x_teste,y_treinamento, y_teste =
train_test_split(previsores,classe,test_size= 0.3, random_state=0)
x_teste


# Criação e treinamento do modelo
naive_bayes = GaussianNB()
naive_bayes.fit(x_treinamento, y_treinamento)


# Previssões utilizando os registros de teste
previsoes = naive_bayes.predict(x_teste)
previsoes


# Geração da matriz de confusão e cálculo da taxa de acerto e erro
confusao = confusion_matrix(y_teste, previsoes)
confusão

#Visualização da matriz de confusão
v = ConfusionMatrix(GaussianNB())
v.fit(x_treinamento,y_treinamento)
v.score(x_teste, y_teste)
v.poof()
```

```
#Taxade acerto
taxa_acerto = accuracy_score(y_teste,previsoes)
taxa_erro = 1 - taxa_acerto
taxa_acerto


# Preparar a nova carga
dfNovaCarga = dfNovaCar-
ga[["runtime","genres","imdb_score","imdb_votes","tmdb_score"]]
dfNovaCarga


# Previsão com novo registro, transforma os atributos categóricos em numéricos.
dfNovaCarga = dfNovaCarga.iloc[:,0:5].values
dfNovaCarga


# Transformação dos atributos categóricos em atributos numéricos, passando o índice de
cada coluna categórica
labelencoder1 = LabelEncoder()
dfNovaCarga[:,1] = labelencoder1.fit_transform(dfNovaCarga[:,1])


#Inserir nova carga no modelo para prever os resultados
result=naive_bayes.predict(dfNovaCarga)
result


# Visualizar o dataset
dfResult


#Adicionar ao final do dataset o resultado das previsões
dfResult['Status'] = result.tolist()


#Visualizar o dataset
dfResult


# Separar as previssões da classificação (Status)
previsores = dataSet.iloc[:,0:5].values
classe = dataSet.iloc[:,5].values


# Transformação dos atributos categóricos em atributos numéricos, passando o índice de
cada coluna categórica
labelencoder2 = LabelEncoder()
previsores[:,1] = labelencoder2.fit_transform(previsores[:,1])
```

```
#Visualizar os valores dos previsores
previsores


#Visualizar os valores das classificações
classe


#Criar as variáveis para treinamento
x_treinamento, x_teste,y_treinamento, y_teste =
train_test_split(previsores,classe,test_size= 0.3, random_state=0)
x_teste


#Criação e treinamento do modelo - Árvore de decisão
arvore = DecisionTreeClassifier(
                        max_depth = None,
                        max_features = None,
                        min_samples_leaf=1,
                        min_samples_split=2
)
arvore.fit(x_treinamento, y_treinamento)


# Exportação da árvore de decisão para o formato .dot, para posterior visualização
export_graphviz(arvore, out_file='tree.dot')
arquivo = 'tree.dot'


# obtenção das previsões
previsoes=arvore.predict(x_teste)
previsoes


# matriz de confusão - mostra os acertos e os erros.
confusao = confusion_matrix(y_teste, previsoes)
confusao


# taxa de acerto - é superior ao modelo de Naive Bayers
taxa_acerto = accuracy_score(y_teste, previsoes)
taxa_acerto


# taxa de erro
taxa_erro = 1 - taxa_acerto
taxa_erro
```